在NuMicro USB產品實現USB複合裝置

Application Note for 32-bit NuMicro™ Family

Document Information

摘要	本文介紹USB複合裝置(Composite Device)的基本概念及實作。
適用範圍	具備USB裝置功能之Cortex™-M0 系列微控制器。

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 概述 3

2 USB 複合裝置(COMPOSITE DEVICE) 4

2.1 介紹 4

2.2 USB 通訊架構 5

2.3 介面(Interface) 6

2.4 端點(Endpoints) 6

2.5 描述元(Descriptor) 9

2.5.1裝置描述元(Device descriptor) 10
2.5.2組態描述元(Configuration descriptor) 11
2.5.3HID特定的類別描述元 14

2.6 驅動程式 18

3 複合裝置範例 22

3.1 應用設計 22

3.2 應用範例開發 35

3.2.1HID Transfer + MSC 35
3.2.2VCOM + HID Transfer 50

3.3 應用範例 66

3.3.1HID Transfer + MSC 67
3.3.2HID Transfer + HID Keyboard 78
3.3.3HID Mouse + HID Keyboard 88
3.3.4VCOM + HID Transfer 98
3.3.5VCOM + MSC 116
3.3.6VCOM + HID Keyboard 127
3.3.7Dual VCOM 139
3.3.8Audio + HID Transfer 158

4 WINDOWS TOOL 178

4.1 概述 178

4.2 Demo 178

1 概述

新唐的Cortex™-M0系列支持USB控制器是一個USB2.0全速設備控制器。它符合USB 2.0全速器件規範，並支持控制/中斷/巨量/等時傳輸類型。請參閱USB Specification Revision 2.0。

本文將介紹USB複合裝置(Composite Device)的基本概念及如何以Nano100B現有的USB類別功能來實作具有兩種功能的複合裝置，USB複合裝置的範例有：

- HID Transfer + MSC
- HID Transfer + HID Keyboard
- HID Mouse + HID Keyboard
- VCOM + HID Transfer
- VCOM + MSC
- VCOM + HID Keyboard
- Dual VCOM
- Audio + HID Transfer

2 USB 複合裝置(Composite Device)

2.1 介紹

複合裝置是一個具有多個功能和配置多個介面的組合設備，但是只有一個唯一的設備(Device)地址，透過相對應的USB設備驅動程式(Device Driver)來與主機通訊。在USB Specification Revision 2.0定義USB複合裝置(Composite Device)為「具有多個獨立控制介面(Interface)的裝置」。

通常一個USB主機呈現一個單一功能的USB設備，一個功能(Function)佔用一個USB纜線(Cable)和USB埠(Port)，例如滑鼠、鍵盤、RS-232或移動儲存裝置。為了同時使用多個功能，需要許多的USB纜線和USB埠，然而複合裝置卻可使用僅有的一條纜線和一個USB埠實現多個功能。

USB設備定義許多不同的類別(Class)，常見的類別有：USB 通訊裝置類別(CDC)、人性化介面裝置類別(HID)、大容量儲存裝置類別(MSC)，以及USB音訊裝置類別(Audio)，使用多個類別功能組合成一個單一的USB埠被稱為複合裝置，例如，鍵盤和滑鼠，USB隨身碟和USB轉RS232等。本文以NANO100B現有的基本USB類別功能來開發兩種功能的複合裝置，表 1列出本文支援的USB類別的示範程式。

複合裝置包含多個功能，但是每個功能可以只有一個(例如：HID或MSC)或更多(例如：CDC或UAC)介面。因此，複合裝置的介面描述元(Interface descriptor)必須小心使用。若是複合裝置中有單一功能對應多個介面的類別就必須額外使用介面關聯描述元(Interface Association Descriptor, IAD)，讓設備中的不同介面與其相同功能建立關聯。本文提供兩種功能的複合裝置範例如下：

非多重介面功能的複合裝置：

- HID Transfer + MSC
- HID Transfer + HID Keyboard
- HID Mouse + HID Keyboard

多重介面功能的複合裝置：

- VCOM + HID Transfer
- VCOM + MSC
- VCOM + HID Keyboard
- Dual VCOM
- Audio + HID Transfer

USB類別	範例	說明
HID	HID Transfer	透過HID介面讓USB設備與電腦之間做資料傳輸，搭配主機端的應用程式來測試此功能。主機端的應用程式請參考Windows Tool。
	HID Keyboard	利用GPIO PB15 pin接地來模擬在鍵盤上輸入按鍵'A'。
	HID Mouse	模擬滑鼠在電腦螢幕上畫圓移動顯示。
MSC	Mass storage	使用晶片內的data flash當作儲存容量的儲存裝置。
CDC	VCOM	在電腦上虛擬出 COM port，測試此功能使用終端機連線工具，需要將設定變更至對應的 COM port 才能連上。
UAC	Audio Speaker	USB audio 音效裝置，透過NAU8822(音訊轉碼器)輸入或輸出音訊資料。

表 1 Nano100 USB 支援的 USB 類別示範程式

2.2 USB 通訊架構

圖 1是主機和裝置之間的通訊方式(詳細內容請參考USB Specification Revision 2.0)。其中主機通常是指電腦，至於裝置，即是常見的周邊裝置設備，但在USB規格書中，稱這種裝置為「功能」(Functions)，其意謂著此系統提供了某些能力，例如具有鍵盤或是滑鼠等功能。通常一個USB設備只提供一個功能給一台USB主機。然而，複合裝置包含多個功能，其中各個功能透過相對應的USB設備驅動程式跟主機做通訊。

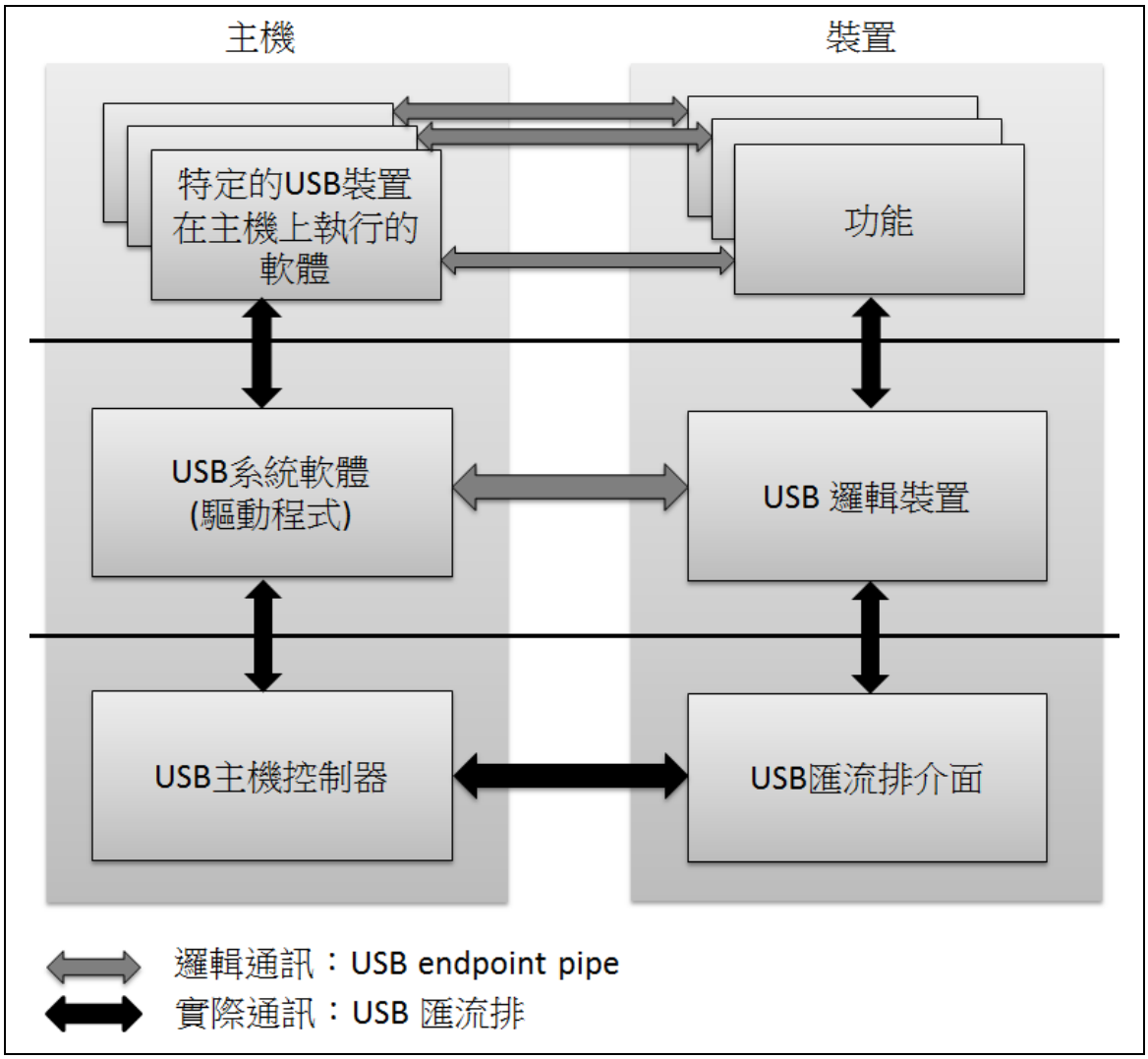


圖 1 USB 通訊方式

2.3 介面(Interface)

複合裝置只有一套PID/VID，透過不同的介面定義為不同的類別來實現多個功能。USB 設備必須額外使用介面關聯描述元(IAD)來定義多重介面功能的複合裝置。複合裝置的介面描述元(Interface Descriptor) 定義介面的每一個功能，USB主機透過此描述元來得知USB設備可用的功能。例如，HID+CDC複合裝置需要三個介面描述元，而HID+MSC複合裝置只需要兩個介面描述元。

2.4 端點(Endpoints)

複合裝置的Endpoint分配，除了控制端點用於標準、特定類別和供應商特定的請求，其他Endpoint是給複合裝置內的個別功能來使用，使用的傳輸方式與應用的需求有關。

Endpoint的數量和Endpoint緩衝區(buffer)大小會根據硬體來定義，實作複合裝置需要仔細安排分配Endpoint。本文件實作使用NANO100B晶片，它提供8個Endpoint及共有512 Byte的緩衝區給各個Endpoint配置使用，每個Endpoint只能選擇一個方向與所指到的資料記憶體儲存空間。表 2與表 3分別整理列出本文所使用的複合裝置範例的介面及Endpoint配置。表 3不同的灰色代表此範例使用一個或兩個IAD描述元來將某些介面關聯組成同一個功能。

	HID Transfer + MSC	**HID Transfer +HID Keyboard**	**HID Mouse + HID Keyboard**
介面0	HID Transfer	HID Transfer	HID Mouse
介面1	MSC	HID Keyboard	HID Keyboard
端點0	Control IN		
端點1	Control OUT		
端點2	HID Interrupt IN	HID Interrupt IN	HID Interrupt IN
端點3	HID Interrupt OUT	HID Interrupt OUT	HID Interrupt IN
端點4	MSC BULK IN	HID Interrupt IN	
端點5	MSC BULK OUT		
端點6			
端點7			

表 2 非多重介面功能複合裝置的介面及端點配置

	VCOM + HID Transfer	VCOM + MSC	VCOM + HID Keyboard	Dual VCOM	Audio + HID Transfer
介面0	VCOM Control	VCOM Control	VCOM Control	VCOM0 Control	Audio Control
介面1	VCOM Data	VCOM Data	VCOM Data	VCOM0 Data	Audio Stream0
介面2	HID Transfer	MSC	HID Keyboard	VCOM1 Control	Audio Stream1
介面3				VCOM1 Data	HID Transfer
端點0	Control IN				
端點1	Control OUT				
端點2	VCOM BULK IN	VCOM BULK IN	VCOM BULK IN	VCOM0 BULK IN	AUDIO ISO IN
端點3	VCOM BULK OUT	VCOM BULK OUT	VCOM BULK OUT	VCOM0 BULK OUT	AUDIO ISO OUT
端點4	VCOM Interrupt IN	VCOM Interrupt IN	VCOM Interrupt IN	VCOM0 Interrupt IN	HID Interrupt IN
端點5	HID Interrupt IN	MSC BULK IN	HID Interrupt IN	VCOM1 Interrupt IN	HID Interrupt OUT
端點6	HID Interrupt OUT	MSC BULK OUT		VCOM1 BULK OUT	
端點7				VCOM1 BULK IN	

表 3 多重介面功能複合裝置的介面及端點配置

2.5 描述元(Descriptor)

每一個裝置都有相關描述元集合，描述元是一種描述裝置跟USB特性的資料結構，用來將裝置的特性呈現給主機。基本的描述元架構如圖2所示，其中包含：裝置描述元（device descriptor）、組態描述元（configuration descriptor）、介面描述元（interface descriptor）、特定的類別描述元(class-specific descriptors)、端點描述元（endpoint descriptor）及字串描述元(String descriptor)。連線到主機的一個裝置僅能有一個裝置描述元，裝置描述元中會交代組態描述元和字串描述元的個數，組態描述元至少要有一個，字串描述元是在電腦主機上查看USB裝置時，看到的各個USB裝置描述，例如：此裝置的用途、公司名字等。一個USB裝置的功能被使用之前，主機會先向USB裝置取得組態資料，然後再由USB裝置將這些描述元的資料當作組態資料一起傳給主機，組態描述元除了本身組態描述元外，另外包含介面描述元、類別描述元、和端點描述元，而HID類別則會有特定的類別描述元，此描述元會交代報告(report)描述元和實體(physical)描述元；一個HID介面至少要有一個報告描述元，報告描述元中描述所要交換的資料，而實體描述元可有可無。

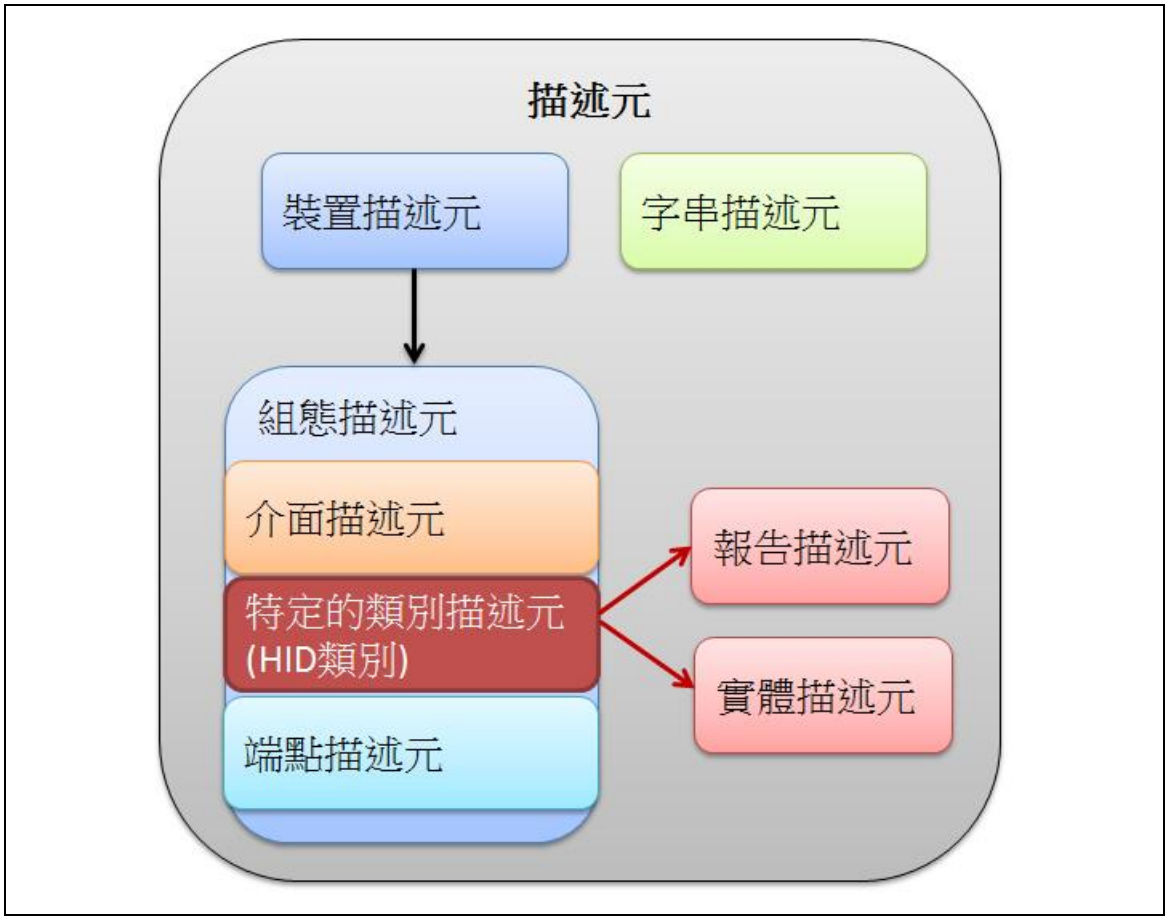


圖 2 描述元的架構

2.5.1 裝置描述元(Device descriptor)

本文所介紹的複合裝置分為兩種類型，一種為非多重介面功能的複合裝置(Composite device without multi-interface functions)，例如：HID Transfer + MSC、HID Transfer + HID Keyboard和HID Mouse + HID Keyboard；另一種則是多重介面功能的複合裝置(Composite device with multi-interface function)，例如：VCOM + HID Transfer、VCOM + MSC、VCOM + HID Keyboard、Dual VCOM和Audio + HID Transfer。圖 3與圖 4分別為非多重介面功能與多重介面功能複合裝置的設備描述元。識別USB設備是否使用IAD是透過設備描述元中的三個欄位(bDeviceClass、bDeviceSubClass及bDeviceProtocol)來判斷，三個欄位的值分別為：0xEF, 0x02, 0x01。

```
/*----------------------------------------------*/
/*!<USB Device Descriptor */
uint8_t gu8DeviceDescriptor[] = {
    LEN_DEVICE,      /* bLength */
    DESC_DEVICE,     /* bDescriptorType */
    0x10, 0x01,      /* bcdUSB */
    0x00,            /* bDeviceClass */
    0x00,            /* bDeviceSubClass */
    0x00,            /* bDeviceProtocol */
    EP0_MAX_PKT_SIZE,    /* bMaxPacketSize0 */
    /* idVendor */
    USBD_VID & 0x00FF,
    (USBD_VID & 0xFF00) >> 8,
    /* idProduct */
    USBD_PID & 0x00FF,
    (USBD_PID & 0xFF00) >> 8,
    0x00, 0x00,      /* bcdDevice */
    0x01,            /* iManufacture */
    0x02,            /* iProduct */
    0x02,            /* iSerialNumber */
    0x01             /* bNumConfigurations */
};
```

圖 3 非多重介面複合裝置的設備描述元

```
/*-----------------------------------------------*/
/*!<USB Device Descriptor */
uint8_t gu8DeviceDescriptor[] = {
    LEN_DEVICE,          /* bLength */
    DESC_DEVICE,         /* bDescriptorType */
    0x00, 0x02,          /* bcdUSB */
    0xEF,                /* bDeviceClass: IAD*/
    0x02,                /* bDeviceSubClass */
    0x01,                /* bDeviceProtocol */
    EP0_MAX_PKT_SIZE,    /* bMaxPacketSize0 */
    /* idVendor */
    USBD_VID & 0x00FF,
    (USBD_VID & 0xFF00) >> 8,
    /* idProduct */
    USBD_PID & 0x00FF,
    (USBD_PID & 0xFF00) >> 8,
    0x00, 0x01,         /* bcdDevice */
    0x01,               /* iManufacture */
    0x02,               /* iProduct */
    0x03,               /* iSerialNumber */
    0x01                /* bNumConfigurations */
};
```

圖 4 多重介面複合裝置的設備描述元

2.5.2 組態描述元(Configuration descriptor)

組態描述元除了本身組態描述元外，另外還包含介面(interface)描述元、類別(class)描述元、和端點(endpoint)描述元。多重介面功能的複合裝置介面描述元就必須使用介面關聯描述元(IAD)，以便將多重介面關聯在單一裝置功能中，可看作是多個獨立的USB設備個別運作。注意，IAD中的第五個欄位bFunctionClass和第六個欄位bFunctionSubclassClass，填寫時需要小心這兩個值要與接下來要被關聯的介面描述元的bInterfaceClass和bInterfaceSubClass相同。這裡利用VCOM+MSC複合裝置的範例來說明此設備的組態描述元的佈局安排。

- 功能 1：VCOM(CDC Class)，透過 IAD 定義相同功能的兩個介面。(介面 0 和介面 1)
- 功能 2：Mass Storage(MSC Class)，這個功能只有一個介面。(介面 2)

VCOM+MSC複合裝置的組態描述元如下：

```
    LEN_CONFIG,       /* bLength              */
    DESC_CONFIG,      /* bDescriptorType      */
    0x62, 0x00,       /* wTotalLength         */
    0x03,             /* bNumInterfaces       */
    0x01,             /* bConfigurationValue  */
    0x00,             /* iConfiguration       */
    0xC0,             /* bmAttributes         */
    0x32,             /* MaxPower             */
```

IAD

```
    // IAD
    0x08,             // bLength: Interface Descriptor size
    0x0B,             // bDescriptorType: IAD
    0x00,             // bFirstInterface
    0x02,             // bInterfaceCount
    0x02,             // bFunctionClass: CDC
    0x02,             // bFunctionSubClass
    0x01,             // bFunctionProtocol
    0x00,             // iFunction
```

介面0描述元 – CDC Control

```
/* VCOM */
    /* INTERFACE descriptor */
    LEN_INTERFACE,    /* bLength              */
    DESC_INTERFACE,   /* bDescriptorType      */
    0x00,             /* bInterfaceNumber     */
    0x00,             /* bAlternateSetting    */
    0x01,             /* bNumEndpoints        */
    0x02,             /* bInterfaceClass      */
    0x02,             /* bInterfaceSubClass   */
    0x01,             /* bInterfaceProtocol   */
    0x00,             /* iInterface           */
```

特定的類別描述元

…

…

…

端點描述元

…

…

…

介面1描述元 – CDC Data

```
/* INTERFACE descriptor */
LEN_INTERFACE,   /* bLength              */
DESC_INTERFACE,  /* bDescriptorType      */
0x01,            /* bInterfaceNumber     */
0x00,            /* bAlternateSetting    */
0x02,            /* bNumEndpoints        */
0x0A,            /* bInterfaceClass      */
0x00,            /* bInterfaceSubClass   */
0x00,            /* bInterfaceProtocol   */
0x00,            /* iInterface           */
```

特定的類別描述元

…

…

…

端點描述元

…

…

…

介面2描述元 – MSC

```
/* MSC class device */
    /* INTERFACE descriptor */
    LEN_INTERFACE,   // bLength
    DESC_INTERFACE,  // bDescriptorType
    0x02,            // bInterfaceNumber
    0x00,            // bAlternateSetting
    0x02,            // bNumEndpoints
    0x08,            // bInterfaceClass
    0x06,            // bInterfaceSubClass
    0x50,            // bInterfaceProtocol
    0x00,            // iInterface
```

特定的類別描述元

…

…

…

端點描述元

…

…

…

2.5.3 HID 特定的類別描述元

HID類別會有特定的類別描述元，它是由描述HID介面的HID描述元及報告描述元所組成。HID描述元是放在組態描述元中，在列舉時用來辨識HID裝置，再經由報告描述元內定義的格式來傳送與接收資料。

以下為HID Mouse加上HID Keyboard的複合裝置範例的組態描述元佈局，其中包含兩個HID描述元。

```
LEN_CONFIG,        /* bLength */
DESC_CONFIG,       /* bDescriptorType */
/* wTotalLength */
0x3B,
0x00,
0x02,              /* bNumInterfaces */
0x01,              /* bConfigurationValue */
0x00,              /* iConfiguration */
/* bmAttributes */
0x80 | (USBD_SELF_POWERED << 6) | (USBD_REMOTE_WAKEUP << 5),
/* MaxPower */
USBD_MAX_POWER,
```

介面0描述元 – 滑鼠

```
/* I/F descr: HID - Mouse */
LEN_INTERFACE,     /* bLength */
DESC_INTERFACE,    /* bDescriptorType */
0x00,              /* bInterfaceNumber */
0x00,              /* bAlternateSetting */
0x01,              /* bNumEndpoints */
0x03,              /* bInterfaceClass */
0x00,              /* bInterfaceSubClass */
HID_MOUSE,         /* bInterfaceProtocol */
0x00,              /* iInterface */
```

HID描述元 – 滑鼠

```
/* HID Descriptor */
LEN_HID,          /* Size of this descriptor in UINT8s. */
DESC_HID,         /* HID descriptor type. */
0x10, 0x01,       /* HID Class Spec. release number. */
0x00,             /* H/W target country. */
0x01,             /* Number of HID class descriptors to follow. */
DESC_HID_RPT,     /* Descriptor type. */
/* Total length of report descriptor. */
sizeof(HID_MouseReportDescriptor) & 0x00FF,
(sizeof(HID_MouseReportDescriptor) & 0xFF00) >> 8,
```

端點描述元

…

…

…

介面1描述元 – 鍵盤

```
/* I/F descr: HID - Keyboard */
LEN_INTERFACE,   /* bLength */
DESC_INTERFACE,  /* bDescriptorType */
0x01,            /* bInterfaceNumber */
0x00,            /* bAlternateSetting */
0x01,            /* bNumEndpoints */
0x03,            /* bInterfaceClass */
0x00,            /* bInterfaceSubClass */
HID_KEYBOARD,    /* bInterfaceProtocol */
0x00,            /* iInterface */
```

HID描述元 – 鍵盤

```
/* HID Descriptor */
LEN_HID,          /* Size of this descriptor in UINT8s. */
DESC_HID,         /* HID descriptor type. */
0x10, 0x01,       /* HID Class Spec. release number. */
0x00,             /* H/W target country. */
0x01,             /* Number of HID class descriptors to follow. */
DESC_HID_RPT,     /* Descriptor type. */
/* Total length of report descriptor. */
sizeof(HID_KeyboardReportDescriptor) & 0x00FF,
(sizeof(HID_KeyboardReportDescriptor) & 0xFF00) >> 8,
```

端點描述元

…

…

…

裝置列舉過程中，取得描述元最為關鍵，下面將介紹裝置列舉時主機向USB設備發出USB GetDescriptor請求來取得各種描述元，由於HID Mouse加上HID Keyboard的複合裝置範例由兩個HID功能組成，所以描述元除了有設備描述元、組態描述元及字串描述元，還另外包含HID描述元和報告描述元。USB GetDescriptor的程式如下：

```
void USBD_GetDescriptor(void)
{
    uint32_t u32Len;

    u32Len = 0;
    u32Len = g_usbd_SetupPacket[7];
    u32Len <<= 8;
    u32Len += g_usbd_SetupPacket[6];

    switch (g_usbd_SetupPacket[3]) {
    // 取得設備描述元
    case DESC_DEVICE: {
        u32Len = Minimum(u32Len, LEN_DEVICE);
        USBD_PrepareCtrlIn((uint8_t *)g_usbd_sInfo->gu8DevDesc, u32Len);
        USBD_PrepareCtrlOut(0,0);
        break;
    }
    // 取得組態描述元
    case DESC_CONFIG: {
        uint32_t u32TotalLen;

        u32TotalLen = g_usbd_sInfo->gu8ConfigDesc[3];
        u32TotalLen = g_usbd_sInfo->gu8ConfigDesc[2] + (u32TotalLen << 8);

        u32Len = Minimum(u32Len, u32TotalLen);
        USBD_PrepareCtrlIn((uint8_t *)g_usbd_sInfo->gu8ConfigDesc, u32Len);
        USBD_PrepareCtrlOut(0,0);
        break;
    }
    // 取得HID描述元
    case DESC_HID: {
        uint32_t u32ConfigDescOffset;
```

```
        u32Len = Minimum(u32Len, LEN_HID);
        u32ConfigDescOffset = g_usbd_sInfo->
gu32ConfigHidDescIdx[g_usbd_SetupPacket[4]];
        USBD_PrepareCtrlIn((uint8_t *)&g_usbd_sInfo->
gu8ConfigDesc[u32ConfigDescOffset], u32Len);
        USBD_PrepareCtrlOut(0,0);
        break;
    }
    // 取得HID報告描述元
    case DESC_HID_RPT: {
        u32Len = Minimum(u32Len, g_usbd_sInfo->
gu32HidReportSize[g_usbd_SetupPacket[4]]);
        USBD_PrepareCtrlIn((uint8_t *)g_usbd_sInfo->
gu8HidReportDesc[g_usbd_SetupPacket[4]], u32Len);
        USBD_PrepareCtrlOut(0,0);
        break;
    }
    // 取得字串描述元
    case DESC_STRING: {
        if(g_usbd_SetupPacket[2] < 4)
        {
            u32Len = Minimum(u32Len, g_usbd_sInfo->
gu8StringDesc[g_usbd_SetupPacket[2]][0]);
            USBD_PrepareCtrlIn((uint8_t *)g_usbd_sInfo->
gu8StringDesc[g_usbd_SetupPacket[2]], u32Len);
            USBD_PrepareCtrlOut(0, 0);
        }
        else
        {
            // Not support. Reply STALL.
            USBD_SET_EP_STALL(EP0);
            USBD_SET_EP_STALL(EP1);
        }
        break;
    }
    default:
        // Not support. Reply STALL.
        USBD_SET_EP_STALL(EP0);
        USBD_SET_EP_STALL(EP1);
        break;
```

```
    }
}
```

若使用的複合裝置有HID功能，在實作時需注意下列三個與HID特定的類別描述元相關的結構，以下程式列出這三個與HID特定的類別描述元的結構。

```
uint8_t *gu8UsbHidReport[3] = {
    HID_MouseReportDescriptor,
    HID_KeyboardReportDescriptor,
    NULL,
};
uint32_t gu32UsbHidReportLen[3] = {
    sizeof(HID_MouseReportDescriptor),
    sizeof(HID_KeyboardReportDescriptor),
    0,
};
uint32_t gu32ConfigHidDescIdx[3] = {
    (LEN_CONFIG+LEN_INTERFACE),
    (sizeof(gu8ConfigDescriptor) - LEN_HID - LEN_ENDPOINT),
    0,
};
```

2.6 驅動程式

當USB複合裝置接上電腦主機時，主機透過安裝驅動程式來辨識USB裝置，這些驅動程式能讓對應的裝置運作，並讓主控端的中央處理器能進行各項應用及操作，當裝置端與主控端的連線正確時，就可進行資料傳輸與通訊等功能。

大部分的電腦作業系統都支援這些裝置類別，為其提供驅動程式。常見的裝置如滑鼠、鍵盤、隨身碟等，都會採用電腦作業系統中的HID或MSC類別驅動程式。Windows載入INF檔案中的usbser.sys來當作CDC通訊設備的驅動程式。INF檔案中必須包含USB裝置的PID及VID，圖 5為VCOM+MSC複合裝置的INF檔案，檔案格式相當簡單，檔案內包含由USB設計論壇(USB-IF)取得授權的公司和應用產品獨有的VID/PID、設備的相關資訊及作業系統在辨認或管理裝置時所使用的設備名稱。為了避免USB設備使用相同PID造成Windows辨識裝置錯誤(Windows會根據VID/PID載入對應的驅動程式)，應用實例中包含CDC通訊設備則會使用不同的PID。表 4列出本文複合裝置的VID/PID。

```
[Version]
Signature  = "$Windows NT$"
Provider   = %COMPANY%
DriverVer  = 10/02/2014,1.0.0.1
CatalogFile   = NuvotonCDC.cat
Class   = Ports
ClassGuid  = {4D36E978-E325-11CE-BFC1-08002BE10318}

[Manufacturer]
%MFGNAME%   = Devices,NT,NTamd64

;-----------------------------------------------------------------------
; Files
;-----------------------------------------------------------------------

[DestinationDirs]
DefaultDestDir   = 12

;-----------------------------------------------------------------------
; Device driver
;-----------------------------------------------------------------------

[NuvotonCDC_DEV.NT]
Include = mdmcpq.inf
CopyFiles  = FakeModemCopyFileSection
AddReg   = NuvotonCDC_DEV.NT.AddReg

[NuvotonCDC_DEV.NT.AddReg]
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,usbser.sys
HKR,,EnumPropPages32,,"MsPorts.dll,SerialPortPropPageProvider"

[NuvotonCDC_DEV.NT.Services]
AddService=usbser, 0x00000002, DriverService
```

```
;-----------------------------------------------------------------------
; Services
;-----------------------------------------------------------------------

[DriverService]
DisplayName = %SERVICE%
ServiceType  = 1
StartType  = 3
ErrorControl  = 1
ServiceBinary    = %12%\usbser.sys

;-----------------------------------------------------------------------
; Devices
;-----------------------------------------------------------------------

[Devices.NT]
%DESCRIPTION%  = NuvotonCDC_DEV, USB\VID_0416&PID_DC01&MI_00

[Devices.NTamd64]
%DESCRIPTION%  = NuvotonCDC_DEV, USB\VID_0416&PID_DC01&MI_00

;-----------------------------------------------------------------------
; Strings
;-----------------------------------------------------------------------

[Strings]
COMPANY  = "Nuvoton Co., Ltd."
MFGNAME  = "www.nuvoton.com"
DESCRIPTION    = "Nuvoton Virtual Com Port"
SERVICE   = "USB RS-232 Emulation Driver"
```

圖 5 CDC+MSC 複合裝置 INF 檔案

	VID	PID
HID Transfer + MSC	0x0416	0x5020
HID Transfer + HID Keyboard	0x0416	0x5020
HID Mouse + HID Keyboard	0x0416	0xC145
VCOM + HID Transfer	0x0416	0xDC00
VCOM + MSC	0x0416	0xDC01
VCOM + HID Keyboard	0x0416	0xDC02
Dual VCOM	0x0416	0x50A1
Audio + HID Transfer	0x0416	0xB00A

表 4 複合裝置的 VID/PID

3 複合裝置範例

3.1 應用設計

本章討論如何以Nano100B現有的基本USB類別功能來開發兩種功能的複合裝置，開發實作各種複合裝置時，使用者自行修改的項目有：Endpoint的配置、定義描述元及回應裝置的標準請求和類別的特定請求，各個範例章節中將針對這三項的修改做說明。測試環境使用NuMicro™ Nano130 開發板或學習板。所有複合裝置應用實例都可在NANO100B系列的BSP\SampleCode\StdDriver找到，表 5和表 6將分別介紹各個複合裝置實作時所使用的檔案，這些檔案分成兩種類型，一個是USB Driver層的核心檔案，使用者通常不需要自行修改；另一種則是使用者開發不同USB類別裝置的應用程式編程介面(API)，對於大多數類別應用，需要修改類別請求和描述元資料，因此，不同類別範例的類別應用層檔案內容會有所不同。

檔案名稱	描述
usbd(.c,.h)	USB Specification Revision所定義的USB協定，包括匯流排事件、列舉和標準請求。

表 5 USB Driver 層的核心檔案

USB 複合裝置範例	檔案名稱	描述
HID Transfer + MSC	descriptors.c	定義此範例所使用到的裝置描述元、配置描述元、字串描述元、HID描述元和報告描述元。
	hid_transfer_And_MSC.c	HID/MSC類別請求和每個端點的事件處理。
	hid_transfer_And_MSC.h	定義HID class request及報告描述元類型。
	Massstorage.h	定義Mass Storage UFI命令。
	Main.c	此範例的主要程式，裡面包含系統、USB裝置各種的初始化及自訂操作。

HID Transfer + HID Keyboard	descriptors.c	定義此範例所使用到的裝置描述元、配置描述元、字串描述元、HID描述元和報告描述元。
	hid_transfer_And_Keyboard.c	HID類別請求和每個端點的事件處理。
	hid_transfer_And_Keyboard.h	定義HID class request及報告描述元類型。
	Main.c	此範例的主要程式，裡面包含系統、USB裝置各種的初始化及利用GPIO PB15 pin接地來模擬在鍵盤上輸入按鍵'A' 的鍵盤功能。
HID Mouse + HID Keyboard	descriptors.c	定義此範例所使用到的裝置描述元、配置描述元、字串描述元、HID描述元和報告描述元。
	hid_mousekeyboard.c	HID類別請求和每個端點的事件處理。
	hid_mousekeyboard.h	定義HID class request及報告描述元類型。
	Main.c	此範例的主要程式，裡面包含系統及USB裝置初始化，利用GPIO PB15 pin接地來模擬在鍵盤上輸入按鍵'A' 的鍵盤功能及滑鼠游標一直在螢幕上畫圓的滑鼠功能。
VCOM + HID Transfer	descriptors.c	定義此範例所使用到的裝置描述元、配置描述

		元、字串描述元、HID描述元和報告描述元。
	VCOM_And_hid_transfer.c	HID/CDC類別請求和每個端點的事件處理。
	VCOM_And_hid_transfer.h	定義HID/CDC class request及報告描述元類型。
	Main.c	此範例的主要程式，裡面包含系統及USB裝置初始化及虛擬成一個COM port傳輸資料。
VCOM + MSC	descriptors.c	定義此範例所使用到的裝置描述元、配置描述元、字串描述元。
	vcom_massStorage.c	CDC/MSC類別請求和每個端點的事件處理。
	vcom_serial.h	定義CDC class request。
	Massstorage.h	定義Mass Storage UFI命令。
	Main.c	此範例的主要程式，裡面包含系統、USB裝置各種的初始化及虛擬成一個COM port傳輸資料。
VCOM + HID Keyboard	descriptors.c	定義此範例所使用到的裝置描述元、配置描述元、字串描述元、HID描述元和報告描述元。
	VCOM_And_hid_keyboard.c	CDC/HID類別請求和每

		個端點的事件處理。
	VCOM_And_hid_keyboard.h	定義HID/CDC class request及報告描述元類型。
	Main.c	此範例的主要程式，裡面包含系統、USB裝置各種的初始化及其他操作功能。(虛擬成一個COM port傳輸資料及利用GPIO PB15 pin接地來模擬在鍵盤上輸入按鍵'A')
Dual VCOM	descriptors.c	定義此範例所使用到的裝置描述元、配置描述元、字串描述元。
	vcom_serial.c	CDC類別請求和每個端點的事件處理。
	vcom_serial.h	定義CDC class request。
	Main.c	此範例的主要程式，裡面包含系統、USB裝置各種的初始化及虛擬成兩個獨立的COM port。
Audio + HID Transfer	descriptors.c	定義此範例所使用到的裝置描述元、配置描述元、字串描述元、HID描述元和報告描述元。
	usbd_audio.c	UAC/HID類別請求和每個端點的事件處理。
	usbd_audio.h	定義UAC/HID class request及報告描述元類

		型。
	Main.c	此範例的主要程式，裡面包含系統及USB裝置各種的初始化。

表 6 USB 類別的應用層檔案

以下將簡單介紹NANO100B的USB韌體程式用法，讓使用者比較容易理解如何開發新的複合裝置。USB韌體程式可以分為核心層和應用層，核心層的USB協定提供一個中斷服務程式來處理所有的USB匯流排事件；應用層包括USB描述元資料、類別請求和端點資料處理。

NANO100B的USB的中斷處理函數，使用者可以透過它了解整個NANO100B的USB處理流程。使用者可以使用中斷或輪詢檢測USB_INTSTS暫存器來監測USB匯流排上的所有事件，像是讀暫存器中的USB_EPSTS和USB_INTSTS來得知是哪類請求、在哪個端點上，並採取必要的回應(EPx_Handler)，讀取USB_BUSSTS獲取USB重置、插拔、恢復或暫停，以下為USB的中斷處理流程：

```
void USBD_IRQHandler(void)
{
    uint32_t u32IntSts = USBD_GET_INT_FLAG();
    uint32_t u32State = USBD_GET_BUS_STATE();

    /* 熱插拔中斷 */
    if(u32IntSts & USBD_INTSTS_FLDET)
    {
        /* 清除熱插拔中斷狀態 */
        USBD_CLR_INT_FLAG(USBD_INTSTS_FLDET);

        if(USBD_IS_ATTACHED())
        {
            /* USB 插入，開啟USB和PHY */
            USBD_ENABLE_USB();
        }
        else
        {
            /* USB 拔除，關閉USB功能 */
            USBD_DISABLE_USB();
        }
    }
```

```
/* USB匯流排中斷 */
if(u32IntSts & USBD_INTSTS_BUS)
{
    /* 清除USB匯流排狀態 */
    USBD_CLR_INT_FLAG(USBD_INTSTS_BUS);

    if(u32State & USBD_STATE_USBRST)
    {
        /* USB匯流排重置 */
        USBD_ENABLE_USB();
        USBD_SwReset();
    }
    if(u32State & USBD_STATE_SUSPEND)
    {
        /* 開啟USB,關閉PHY */
        USBD_DISABLE_PHY();
    }
    if(u32State & USBD_STATE_RESUME)
    {
        /* 開啟 USB 和 PHY */
        USBD_ENABLE_USB();
    }
}

/* USB事件中斷 */
if(u32IntSts & USBD_INTSTS_USB)
{
    /* SETUP事件 */
    if(u32IntSts & USBD_INTSTS_SETUP)
    {
        /* 清除SETUP事件狀態 */
        USBD_CLR_INT_FLAG(USBD_INTSTS_SETUP);

        /* 清除控制端點的Ready狀態 */
        USBD_STOP_TRANSACTION(EP0);
        USBD_STOP_TRANSACTION(EP1);
        /* 分析Setup Packet */
        USBD_ProcessSetupPacket();
```

```
        }

        /* 端點事件 */
        if(u32IntSts & USBD_INTSTS_EP0) /* EP0被配置為控制端點IN */
        {
            /* 清除EP0事件狀態 */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP0);
            /* 處理控制端點發送數據給主機完完成中斷 */
            USBD_CtrlIn();
        }

        if(u32IntSts & USBD_INTSTS_EP1) /* EP1被配置為控制端點OUT */
        {
            /* 清除EP1事件狀態 */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP1);

            /* 處理主機通過控制端點發送數據給Device */
            USBD_CtrlOut();
        }

        if(u32IntSts & USBD_INTSTS_EP2)
        {
            /* 清除EP2事件狀態 */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP2);
            /* EP2中斷處理函數 */
            EP2_Handler();
        }

        if(u32IntSts & USBD_INTSTS_EP3)
        {
            /* 清除EP3事件狀態 */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP3);
            /* EP3中斷處理函數 */
            EP3_Handler();
        }

        /* 其它端點也可以依照需求，設定端點處理函數 */
        if(u32IntSts & USBD_INTSTS_EP4)
        {
```

```
            /* 清除EP4事件狀態 */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP4);
        }

        if(u32IntSts & USBD_INTSTS_EP5)
        {
            /* 清除EP5事件狀態 */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP5);
        }

        if(u32IntSts & USBD_INTSTS_EP6)
        {
            /* 清除EP6事件狀態 */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP6);
        }

        if(u32IntSts & USBD_INTSTS_EP7)
        {
            /* 清除EP7事件狀態 */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP7);
        }
    }
}
```

NANO100B提供8個Endpoint可以配置成Control/Bulk/Interrupt/Isochronous傳輸類型。一個USB設備中，可以有多個端點，每個端點的作用不同，所以需要端點地址來辨識端點。端點地址定義在USB_CFGx暫存器中的EP_NUM，它用來記錄端點的地址，此地址需要跟端點描述元中的地址相對應，這樣硬體才知道要跟哪個端點通訊，而每個端點發送/接收到的最大封包長度定義在USBD_MXPLD暫存器。以下的程式碼擷取自HID Transfer+MSC複合裝置的範例，用來說明端點地址與端點描述元中的地址關係。其中，HID_Init()用來配置所有被使用的端點，然而端點也會被記錄在組態描述元中的端點描述元裡面，因此在端點描述元裡的端點地址必須根據在HID_Init()所定義的各端點地址來填寫。

```
void HID_Init(void)
{
    /* Init setup packet buffer */
    /* Buffer range for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;
```

```
/*****************************************************/
/* EP0 ==> control IN endpoint, address 0 */
USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN);
/* Buffer range for EP0 */
USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

/* EP1 ==> control OUT endpoint, address 0 */
USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT);
/* Buffer range for EP1 */
USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

/*****************************************************/
/* EP2 ==> Interrupt IN endpoint, address 1 */
USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
/* Buffer range for EP2 */
USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

/* EP3 ==> Interrupt OUT endpoint, address 2 */
USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_OUT | INT_OUT_EP_NUM);
/* Buffer range for EP3 */
USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);
/* trigger to receive OUT data */
USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);

/*****************************************************/
/* EP4 ==> Bulk IN endpoint, address 3 */
USBD_CONFIG_EP(EP4, USBD_CFG_EPMODE_IN | BULK_IN_EP_NUM);
/* Buffer range for EP4 */
USBD_SET_EP_BUF_ADDR(EP4, EP4_BUF_BASE);

/* EP5 ==> Bulk Out endpoint, address 4 */
USBD_CONFIG_EP(EP5, USBD_CFG_EPMODE_OUT | BULK_OUT_EP_NUM);
/* Buffer range for EP5 */
USBD_SET_EP_BUF_ADDR(EP5, EP5_BUF_BASE);
/* trigger to receive OUT data */
USBD_SET_PAYLOAD_LEN(EP5, EP5_MAX_PKT_SIZE);

/*****************************************************/
g_u32BulkBuf0 = EP5_BUF_BASE;
```

```
    g_u32BulkBuf1 = EP4_BUF_BASE;

    g_sCSW.dCSWSignature = CSW_SIGNATURE;
    g_TotalSectors = DATA_FLASH_STORAGE_SIZE / UDC_SECTOR_SIZE;
}

/* USB 組態描述元，它包含端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
    LEN_CONFIG,     /* bLength */
    DESC_CONFIG,    /* bDescriptorType */
    /* wTotalLength */
    0x40,
    0x00,
    0x02,           /* bNumInterfaces */
    0x01,           /* bConfigurationValue */
    0x00,           /* iConfiguration */
    0x80 | (USBD_SELF_POWERED << 6) | (USBD_REMOTE_WAKEUP << 5),/*
bmAttributes */
    USBD_MAX_POWER, /* MaxPower */

    /* I/F descr: HID */
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
    0x00,           /* bInterfaceNumber */
    0x00,           /* bAlternateSetting */
    0x02,           /* bNumEndpoints */
    0x03,           /* bInterfaceClass */
    0x00,           /* bInterfaceSubClass */
    0x00,           /* bInterfaceProtocol */
    0x00,           /* iInterface */

    /* HID Descriptor */
    LEN_HID,        /* Size of this descriptor in UINT8s. */
    DESC_HID,       /* HID descriptor type. */
    0x10, 0x01,     /* HID Class Spec. release number. */
    0x00,           /* H/W target country. */
    0x01,           /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,   /* Descriptor type. */
    /* Total length of report descriptor. */
```

```
    sizeof(HID_DeviceReportDescriptor) & 0x00FF,
    (sizeof(HID_DeviceReportDescriptor) & 0xFF00) >> 8,

    /* EP Descriptor: interrupt in. */
    LEN_ENDPOINT,                                   /* bLength */
    DESC_ENDPOINT,                                  /* bDescriptorType */
    (INT_IN_EP_NUM | EP_INPUT),                     /* bEndpointAddress */
    EP_INT,                                         /* bmAttributes */
    /* wMaxPacketSize */
    EP2_MAX_PKT_SIZE & 0x00FF,
    (EP2_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,                    /* bInterval */

    /* EP Descriptor: interrupt out. */
    LEN_ENDPOINT,                                   /* bLength */
    DESC_ENDPOINT,                                  /* bDescriptorType */
    (INT_OUT_EP_NUM | EP_OUTPUT),                   /* bEndpointAddress */
    EP_INT,                                         /* bmAttributes */
    /* wMaxPacketSize */
    EP3_MAX_PKT_SIZE & 0x00FF,
    (EP3_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,                    /* bInterval */

    /* MSC Descriptor */
    /* const BYTE cbyInterfaceDescriptor[LEN_INTERFACE] = */
    LEN_INTERFACE,                                  // bLength
    DESC_INTERFACE,                                 // bDescriptorType
    0x01,                                           // bInterfaceNumber
    0x00,                                           // bAlternateSetting
    0x02,                                           // bNumEndpoints
    0x08,                                           // bInterfaceClass
    0x06,                                           // bInterfaceSubClass
    0x50,                                           // bInterfaceProtocol
    0x00,                                           // iInterface

    /* const BYTE cbyEndpointDescriptor1[LEN_ENDPOINT] = */
    LEN_ENDPOINT,                                   // bLength
    DESC_ENDPOINT,                                  // bDescriptorType
    (BULK_IN_EP_NUM | EP_INPUT),                    // bEndpointAddress
```

```
    EP_BULK,                                    // bmAttributes
    EP4_MAX_PKT_SIZE, 0x00,                     // wMaxPacketSize
    0x00,                                       // bInterval

    /* const BYTE cbyEndpointDescriptor2[LEN_ENDPOINT] = */
    LEN_ENDPOINT,                               // bLength
    DESC_ENDPOINT,                              // bDescriptorType
    (BULK_OUT_EP_NUM | EP_OUTPUT),              // bEndpointAddress
    EP_BULK,                                    // bmAttributes
    EP5_MAX_PKT_SIZE, 0x00,                     // wMaxPacketSize
    0x00                                        // bInterval
};
```

NANO100B USB的控制器中有512 Bytes SRAM，8個端點共享此Buffer。在功能使用前，使用者必須在USB_BUFSEGx暫存器中填入偏移位址代表配置此端點的有效起始地址，端點發送/接收的長度定義在USB_MXPLDx暫存器。USB Buffer從位址0開始依序分配給SETUP buffer、控制端點IN buffer、控制端點OUT buffer，接下來才分配給其它端點使用。下圖描述依據BUFSEG和MXPLD暫存器內容定義的每個端點起始位址。若BUFSEG0被設定為0x08h，MXPLD0 設定為 0x40h，則端點0的Buffer使用位址是從USB_BASE+0x108h開始，結束位址為USB_BASE+0x147h；其它端點的起始位址依此類推。

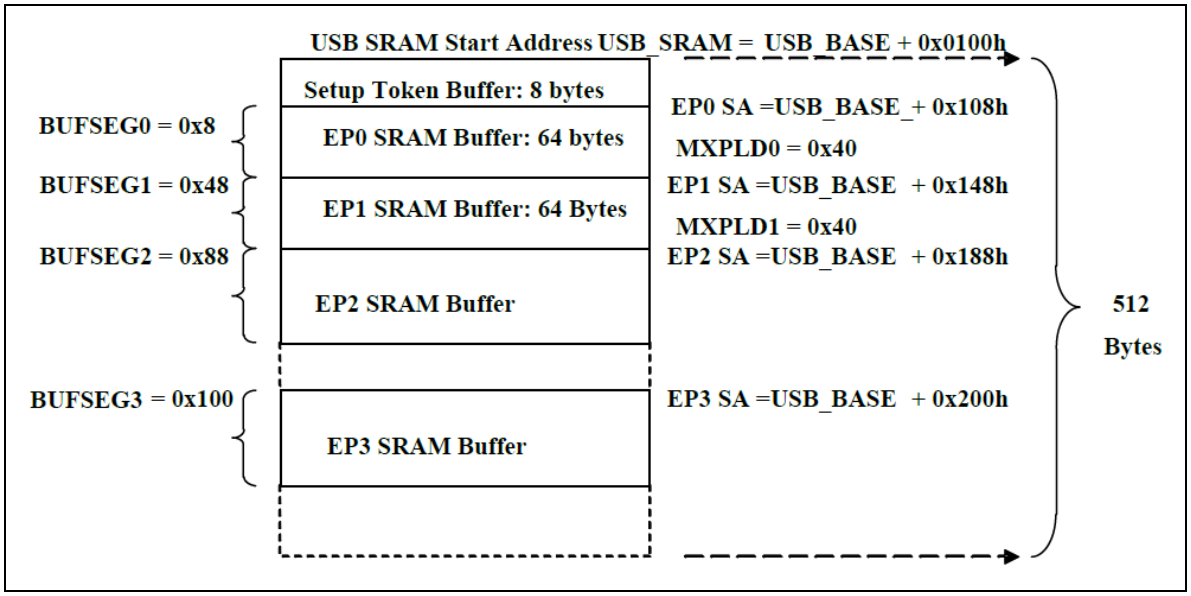


圖 6 USB Buffer 各端點的分配

USB列舉過程中的命令處理，在usbd.c中有完整的處理函數。列舉時主機會向設備要各種描述元，要識別USB設備靠的就是各種USB描述元，在descriptor.c中定

義了USB描述元結構，它包含設備描述元、組態描述元、字串描述元以及HID類別所要使用的報告描述元、報告描述元的長度及HID描述元在組態描述元內的位置，描述元結構如下所示。若是設備有HID功能才需要使用gu8HidReportDesc、gu32HidReportSize及gu32ConfigHidDescIdx。

```
/* 描述元 */
typedef struct s_usbd_info {
    uint8_t *gu8DevDesc;                /* 設備描述元 */
    uint8_t *gu8ConfigDesc;             /* 組態描述元 */
    uint8_t **gu8StringDesc;            /* 字串描述元 */
    uint8_t **gu8HidReportDesc;         /* 報告描述元 */
    uint32_t *gu32HidReportSize;        /* 報告描述元的長度 */
    uint32_t *gu32ConfigHidDescIdx;     /* HID描述元在組態描述元內的位置 */
} S_USBD_INFO_T;

/* 字串描述元 */
uint8_t *gpu8UsbString[4] = {
    gu8StringLang,
    gu8VendorStringDesc,
    gu8ProductStringDesc,
    NULL,
};

/* 報告描述元 */
uint8_t *gu8UsbHidReport[3] = {
    HID_DeviceReportDescriptor,
    NULL,
    NULL,
};

/* 報告描述元的長度 */
uint32_t gu32UsbHidReportLen[3] = {
    sizeof(HID_DeviceReportDescriptor),
    0,
    0,
};

/* HID描述元在組態描述元內的位置 */
uint32_t gu32ConfigHidDescIdx[3] = {
```

```
    (LEN_CONFIG+LEN_INTERFACE),
    0,
    0,
};
```

3.2 應用範例開發

本章節將介紹如何使用NANO100B現有的USB類別功能來開發具有兩種功能的複合裝置的實作過程，著重於告訴使用者如何將兩個獨立的USB類別功能程式碼合併成複合裝置。本文共有八個複合裝置的範例，由於實作方式大同小異，因此，只使用其中兩個範例來說明示範，範例如下：

1. 非多重介面功能的複合裝置：HID Transfer + MSC

2. 多重介面功能的複合裝置：VCOM + HID Transfer

3.2.1 HID Transfer + MSC

從BSP\SampleCode\StdDriver分別取得HID Transfer和MSC這兩個獨立功能的範例程式，使用者可以從中選擇某一個類別範例程式再結合另一個範例程式來實作出新的複合裝置，如此一來可以減少開發的時間。由於MSC的程式碼較繁瑣，因此選擇用MSC範例程式為基底來與HID Transfer範例合併。使用者必須更動以下這些檔案來合併類別範例實作複合裝置。

- classname.c/classname.h：用來處理類別請求和每個端點的事件。classname.c 的檔案會依照類別的名稱來命名。例如裝置為 MSC 類別，則會看到 MassStorage.c；CDC 類別則為 vcom_serial.c。
- descriptor.c：定義範例中所使用到的描述元。

為了方便說明，將程式修改合併步驟分成下面五項：

步驟 1：端點配置及端點初始化

請參考表 2表 2 非多重介面功能複合裝置的介面及端點配置，在USBD_Mass_Storage_Flash\massstorage.h新增HID Transfer所要使用的端點配置，程式如下：

```
/* 端點最大封包長度 */
#define EP0_MAX_PKT_SIZE    8
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    64
#define EP3_MAX_PKT_SIZE    64
#define EP4_MAX_PKT_SIZE    64
#define EP5_MAX_PKT_SIZE    64

/* 各個端點在USB Buffer的偏移值 */
```

```
#define SETUP_BUF_BASE  0
#define SETUP_BUF_LEN   8
#define EP0_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP0_BUF_LEN     EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP1_BUF_LEN     EP1_MAX_PKT_SIZE
#define EP2_BUF_BASE    (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN     EP2_MAX_PKT_SIZE
#define EP3_BUF_BASE    (EP2_BUF_BASE + EP2_BUF_LEN)
#define EP3_BUF_LEN     EP3_MAX_PKT_SIZE
#define EP4_BUF_BASE    (EP3_BUF_BASE + EP3_BUF_LEN)
#define EP4_BUF_LEN     EP4_MAX_PKT_SIZE
#define EP5_BUF_BASE    (EP4_BUF_BASE + EP4_BUF_LEN)
#define EP5_BUF_LEN     EP5_MAX_PKT_SIZE

/* 端點地址 */
#define INT_IN_EP_NUM       0x01
#define INT_OUT_EP_NUM      0x02
#define BULK_IN_EP_NUM      0x03
#define BULK_OUT_EP_NUM     0x04
```

端點配置完之後，在USBD_Mass_Storage_Flash\MassStorage.c中的MSC_Init()裡面新增HID的端點初始化，並且把MSC的端點改成EP4和EP5。

```
void MSC_Init(void)
{
    /* Init setup packet buffer */
    /* Buffer range for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;

    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);
```

```
    /*****************************************************/
    /* EP2 ==> Interrupt IN endpoint, address 1 */
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
    /* Buffer range for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

    /* EP3 ==> Interrupt OUT endpoint, address 2 */
    USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_OUT | INT_OUT_EP_NUM);
    /* Buffer range for EP3 */
    USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);
    /* trigger to receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);

    /*****************************************************/
    /* EP4 ==> Bulk IN endpoint, address 3 */
    USBD_CONFIG_EP(EP4, USBD_CFG_EPMODE_IN | BULK_IN_EP_NUM);
    /* Buffer range for EP4 */
    USBD_SET_EP_BUF_ADDR(EP4, EP4_BUF_BASE);

    /* EP5 ==> Bulk Out endpoint, address 4 */
    USBD_CONFIG_EP(EP5, USBD_CFG_EPMODE_OUT | BULK_OUT_EP_NUM);
    /* Buffer range for EP5 */
    USBD_SET_EP_BUF_ADDR(EP5, EP5_BUF_BASE);
    /* trigger to receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP5, EP5_MAX_PKT_SIZE);

    /*****************************************************/
    g_u32BulkBuf0 = EP5_BUF_BASE;
    g_u32BulkBuf1 = EP4_BUF_BASE;

    g_sCSW.dCSWSignature = CSW_SIGNATURE;
    g_TotalSectors = DATA_FLASH_STORAGE_SIZE / UDC_SECTOR_SIZE;
}
```

步驟 2：定義描述元

在USBD_Mass_Storage_Flash\descriptor.c新增HID類別特定的類別描述元及報告描述元，並且在組態描述元裡面新增HID裝置的介面描述元及其使用的端點描述元。在這裡需要注意wTotalLength、bNumInterfaces及bInterfaceNumber。其中，

WTotalLength則為組態描述元的總長度。另外，這個範例使用兩個介面，因此，bNumInterfaces的值為2，然而在介面描述元中HID介面的bInterfaceNumber為0，MSC介面的bInterfaceNumber為1。

```
/* USB 組態描述元，它包含介面、HID及端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
/* 組態描述元 */
    LEN_CONFIG,     /* bLength */
    DESC_CONFIG,    /* bDescriptorType */

    /* wTotalLength = LEN_CONFIG+LEN_INTERFACE*2+LEN_HID+LEN_ENDPOINT*4 */
    0x40, 0x00,     /* wTotalLength */
    0x02,           /* bNumInterfaces */
    0x01,           /* bConfigurationValue */
    0x00,           /* iConfiguration */
    0x80 | (USBD_SELF_POWERED << 6) | (USBD_REMOTE_WAKEUP << 5),/*
bmAttributes */
    USBD_MAX_POWER, /* MaxPower */

/* 介面0 : HID Transfer介面描述元*/
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
    0x00,           /* bInterfaceNumber */
    0x00,           /* bAlternateSetting */
    0x02,           /* bNumEndpoints */
    0x03,           /* bInterfaceClass */
    0x00,           /* bInterfaceSubClass */
    0x00,           /* bInterfaceProtocol */
    0x00,           /* iInterface */

/* HID描述元 */
    LEN_HID,        /* Size of this descriptor in UINT8s. */
    DESC_HID,       /* HID descriptor type. */
    0x10, 0x01,     /* HID Class Spec. release number. */
    0x00,           /* H/W target country. */
    0x01,           /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,   /* Descriptor type. */
    /* Total length of report descriptor. */
    sizeof(HID_DeviceReportDescriptor) & 0x00FF,
    (sizeof(HID_DeviceReportDescriptor) & 0xFF00) >> 8,
```

```
/* 端點2描述元: 中斷IN，地址為INT_IN_EP_NUM */
    LEN_ENDPOINT,                                   /* bLength */
    DESC_ENDPOINT,                                  /* bDescriptorType */
    (INT_IN_EP_NUM | EP_INPUT),                     /* bEndpointAddress */
    EP_INT,                                         /* bmAttributes */
    /* wMaxPacketSize */
    EP2_MAX_PKT_SIZE & 0x00FF,
    (EP2_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,                    /* bInterval */

/* 端點3描述元: 中斷OUT，地址為INT_OUT_EP_NUM */
    LEN_ENDPOINT,                                   /* bLength */
    DESC_ENDPOINT,                                  /* bDescriptorType */
    (INT_OUT_EP_NUM | EP_OUTPUT),                   /* bEndpointAddress */
    EP_INT,                                         /* bmAttributes */
    /* wMaxPacketSize */
    EP3_MAX_PKT_SIZE & 0x00FF,
    (EP3_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,                    /* bInterval */

/* 介面1：MSC 介面描述元 */
    LEN_INTERFACE,                                  // bLength
    DESC_INTERFACE,                                 // bDescriptorType
    0x01,                                           // bInterfaceNumber
    0x00,                                           // bAlternateSetting
    0x02,                                           // bNumEndpoints
    0x08,                                           // bInterfaceClass
    0x06,                                           // bInterfaceSubClass
    0x50,                                           // bInterfaceProtocol
    0x00,                                           // iInterface

/* 端點4描述元: BULK IN，地址為BULK_IN_EP_NUM */
    LEN_ENDPOINT,                                   // bLength
    DESC_ENDPOINT,                                  // bDescriptorType
    (EP_INPUT | BULK_IN_EP_NUM),                    // bEndpointAddress
    EP_BULK,                                        // bmAttributes
    EP4_MAX_PKT_SIZE, 0x00,                         // wMaxPacketSize
    0x00,                                           // bInterval
```

```
/* 端點5描述元: BULK OUT，地址為BULK_OUT_EP_NUM */
    LEN_ENDPOINT,                                   // bLength
    DESC_ENDPOINT,                                  // bDescriptorType
    (EP_OUTPUT | BULK_OUT_EP_NUM),                  // bEndpointAddress
    EP_BULK,                                        // bmAttributes
    EP5_MAX_PKT_SIZE, 0x00,                         // wMaxPacketSize
    0x00                                            // bInterval
};

/* HID Transfer報告描述元 */
uint8_t HID_DeviceReportDescriptor[] =
{
    0x06, 0x00, 0xFF,   // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,         // Usage (Vendor Usage 1)
    0xA1, 0x01,         // Collection (Application)
    0x19, 0x01,         // Usage Minimum
    0x29, 0x40,         // Usage Maximum //64 input usages total (0x01 to
0x40)
    0x15, 0x00,         // Logical Minimum (data bytes in the report may have
minimum value = 0x00)
    0x26, 0xFF, 0x00,   // Logical Maximum (data bytes in the report may have
maximum value = 0x00FF = unsigned 255)
    0x75, 0x08,         // Report Size: 8-bit field size
    0x95, 0x40,         // Report Count: Make sixty-four 8-bit fields (the
next time the parser hits
                        // an "Input", "Output", or "Feature" item)
    0x81, 0x00,         // Input (Data, Array, Abs): Instantiates input packet
fields based on the
                        // above report size, count, logical min/max, and
usage.
    0x19, 0x01,         // Usage Minimum
    0x29, 0x40,         // Usage Maximum //64 output usages total (0x01 to
0x40)
    0x91, 0x00,         // Output (Data, Array, Abs): Instantiates output
packet fields. Uses same
                        // report size and count as "Input" fields, since
nothing new/different was
                        // specified to the parser since the "Input" item.
    0xC0                // End Collection
};
```

因為此複合裝置的介面0為HID功能，所以在實作時新增與HID特定的類別描述元相關的結構。USBD_Mass_Storage_Flash\descriptor.c修改程式如下所示。

```
uint8_t *gu8UsbHidReport[3] = {
    HID_DeviceReportDescriptor,            /* HID transfer報告描述元 */
    NULL,
    NULL,
};

uint32_t gu32UsbHidReportLen[3] = {
    sizeof(HID_DeviceReportDescriptor),    /* HID transfer報告描述元的長度 */
    0,
    0,
};

uint32_t gu32ConfigHidDescIdx[3] = {
    (LEN_CONFIG+LEN_INTERFACE),     /* HID transfer描述元在組態描述元內的位置*/
    0,
    0,
};
```

步驟 3：定義類別的特定請求

新增HID設備的六個類別請求，直接從USBD_HID_Transfer\hid_transfer.h把HID的六個請求定義複製到USBD_Mass_Storage_Flash\massstorage.h。

```
/*!<Define Mass Storage Class Specific Request */
#define BULK_ONLY_MASS_STORAGE_RESET    0xFF
#define GET_MAX_LUN                     0xFE

/*!<Define Mass Storage Signature */
#define CBW_SIGNATURE       0x43425355
#define CSW_SIGNATURE       0x53425355

/*!<Define Mass Storage UFI Command */
#define UFI_TEST_UNIT_READY                     0x00
#define UFI_REQUEST_SENSE                       0x03
#define UFI_INQUIRY                             0x12
#define UFI_MODE_SELECT_6                       0x15
#define UFI_MODE_SENSE_6                        0x1A
#define UFI_START_STOP                          0x1B
```

```
#define UFI_PREVENT_ALLOW_MEDIUM_REMOVAL        0x1E
#define UFI_READ_FORMAT_CAPACITY                0x23
#define UFI_READ_CAPACITY                       0x25
#define UFI_READ_10                             0x28
#define UFI_READ_12                             0xA8
#define UFI_READ_16                             0x9E
#define UFI_WRITE_10                            0x2A
#define UFI_WRITE_12                            0xAA
#define UFI_VERIFY_10                           0x2F
#define UFI_MODE_SELECT_10                      0x55
#define UFI_MODE_SENSE_10                       0x5A

/*----------------------------------------*/
#define BULK_CBW  0x00
#define BULK_IN   0x01
#define BULK_OUT  0x02
#define BULK_CSW  0x04
#define BULK_NORMAL 0xFF

/*!<Define HID Class Specific Request */
#define GET_REPORT          0x01
#define GET_IDLE            0x02
#define GET_PROTOCOL        0x03
#define SET_REPORT          0x09
#define SET_IDLE            0x0A
#define SET_PROTOCOL        0x0B

/*!<USB HID Interface Class protocol */
#define HID_NONE            0x00
#define HID_KEYBOARD        0x01
#define HID_MOUSE           0x02

/*!<USB HID Class Report Type */
#define HID_RPT_TYPE_INPUT      0x01
#define HID_RPT_TYPE_OUTPUT     0x02
#define HID_RPT_TYPE_FEATURE    0x03
```

USBD_Mass_Storage_Flash\massstorage.h請參考表 4修改成此複合裝置所對應的VID/PID。

```
#define USBD_VID        0x0416
#define USBD_PID        0x5020
```

步驟 4：回應類別的特定請求

USBD_Mass_Storage_Flash\MassStorage.c已經有回應MSC的兩個類別請求，只要在這個檔案加入回應HID設備所需要的六個類別請求。使用者可以從USBD_HID_Transfer\hid_transfer.c複製回應HID的六個請求的程式碼，再依照步驟1的端點配置修改為相對應的地址及傳輸類型。

```
void MSC_ClassRequest(void)
{
    uint8_t buf[8];

    USBD_GetSetupPacket(buf);

    if (buf[0] & 0x80) { /* request data transfer direction */
        // Device to host
        switch (buf[1]) {
        case GET_MAX_LUN: {
            if ((((buf[3]<<8)+buf[2]) == 0) && (((buf[5]<<8)+buf[4]) == 0) &&
(((buf[7]<<8)+buf[6]) == 1))
           {
                M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = 0;
                /* Data stage */
                USBD_SET_DATA1(EP0);
                USBD_SET_PAYLOAD_LEN(EP0, 1);
                /* Status stage */
                USBD_PrepareCtrlOut(0,0);
            }
            else {/* Invalid Get MaxLun command */
                USBD_SetStall(0);
            }
            break;
        }
        case GET_REPORT:
//              {
//                  break;
//              }
        case GET_IDLE:
//              {
```

```
//                  break;
//              }
        case GET_PROTOCOL:
//            {
//                break;
//            }
        default: {
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
        }
    } else {
        // Host to device
        switch (buf[1]) {
        case SET_REPORT: {
            if (buf[3] == 3) {
                /* Request Type = Feature */
                USBD_SET_DATA1(EP1);
                USBD_SET_PAYLOAD_LEN(EP1, 0);
            }
            break;
        }
        case SET_IDLE: {
            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;
        }
        case BULK_ONLY_MASS_STORAGE_RESET: {
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);

            USBD_LockEpStall(0);

            /* Clear ready */
            USBD->EP[EP4].CFG |= USBD_CFG_CLRRDY_Msk;
            USBD->EP[EP5].CFG |= USBD_CFG_CLRRDY_Msk;
```

```
            /* Prepare to receive the CBW */
            g_u8EP5Ready = 0;
            g_u8BulkState = BULK_CBW;
            USBD_SET_DATA1(EP5);
            USBD_SET_EP_BUF_ADDR(EP5, g_u32BulkBuf0);
            USBD_SET_PAYLOAD_LEN(EP5, EP5_MAX_PKT_SIZE);
            break;
        }
        case SET_PROTOCOL:
//              {
//                  break;
//              }
        default: {
            // Stall
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
        }
    }
```

步驟 5：處理每個端點發生的事件

USBD_Mass_Storage_Flash\MassStorage.c中的USBD_IRQHandler()得知哪個端點發生中斷，並且執行該端點的Callback function。使用者在USBD_IRQHandler()中新增HID端點2和端點3的Callback function，並且修改MSC端點的Callback function為EP4_Handler和EP5_Handler。

```
void USBD_IRQHandler(void)
{
    uint32_t u32IntSts = USBD_GET_INT_FLAG();
    uint32_t u32State = USBD_GET_BUS_STATE();

//-----------------------------------------------------------------
    if (u32IntSts & USBD_INTSTS_FLDET) {
        // Floating detect
        USBD_CLR_INT_FLAG(USBD_INTSTS_FLDET);

        if (USBD_IS_ATTACHED()) {
            /* USB Plug In */
            USBD_ENABLE_USB();
```

```
        } else {
            /* USB Un-plug */
            USBD_DISABLE_USB();
        }
    }

//-----------------------------------------------------------------
    if (u32IntSts & USBD_INTSTS_BUS) {
        /* Clear event flag */
        USBD_CLR_INT_FLAG(USBD_INTSTS_BUS);

        if (u32State & USBD_STATE_USBRST) {
            /* Bus reset */
            USBD_ENABLE_USB();
            USBD_SwReset();
            g_u8Remove = 0;
        }
        if (u32State & USBD_STATE_SUSPEND) {
            /* Enable USB but disable PHY */
            USBD_DISABLE_PHY();
        }
        if (u32State & USBD_STATE_RESUME) {
            /* Enable USB and enable PHY */
            USBD_ENABLE_USB();
        }
    }

//-----------------------------------------------------------------
    if (u32IntSts & USBD_INTSTS_USB) {
        // USB event
        if (u32IntSts & USBD_INTSTS_SETUP) {
            // Setup packet
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_SETUP);

            /* Clear the data IN/OUT ready flag of control end-points */
            USBD_STOP_TRANSACTION(EP0);
            USBD_STOP_TRANSACTION(EP1);
```

```
            USBD_ProcessSetupPacket();
        }

        // EP events
        if (u32IntSts & USBD_INTSTS_EP0) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP0);
            // control IN
            USBD_CtrlIn();
        }

        if (u32IntSts & USBD_INTSTS_EP1) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP1);
            // control OUT
            USBD_CtrlOut();
        }

        if (u32IntSts & USBD_INTSTS_EP2) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP2);
            // Interrupt IN
            EP2_Handler();
        }

        if (u32IntSts & USBD_INTSTS_EP3) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP3);
            // Interrupt OUT
            EP3_Handler();
        }

        if (u32IntSts & USBD_INTSTS_EP4) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP4);
            // Bulk IN
            EP4_Handler();
        }
```

```
        if (u32IntSts & USBD_INTSTS_EP5) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP5);
            // Bulk OUT
            EP5_Handler();
        }

        if (u32IntSts & USBD_INTSTS_EP6) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP6);
        }

        if (u32IntSts & USBD_INTSTS_EP7) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP7);
        }
    }
    /* clear unknown event */
    USBD_CLR_INT_FLAG(u32IntSts);
}

void EP2_Handler(void)  /* Interrupt IN handler */
{
    HID_SetInReport();
}

void EP3_Handler(void)  /* Interrupt OUT handler */
{
    uint8_t *ptr;
    /* Interrupt OUT */
    ptr = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP3));
    HID_GetOutReport(ptr, USBD_GET_PAYLOAD_LEN(EP3));
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
}

void EP4_Handler(void)
{
    g_u8EP4Ready = 1;
    MSC_AckCmd();
```

```
}

void EP5_Handler(void)
{
    /* Bulk OUT */
    g_u8EP5Ready = 1;
}
```

最後，把USBD_HID_Transfer\hid_transfer.c裡面HID相關的端點處理程式複製至USBD_Mass_Storage_Flash\MassStorage.c，這樣就完成HID Transfer+MSC的複合裝置。然而，這裡要特別注意每個裝置所對應的端點處理程式，程式必須根據其對應的端點來做修改，否則會無法成功辨識此裝置，造成裝置無法使用。下面的程式碼以MSC_Read的命令處理來簡單說明。這個範例把原本MSC的端點從EP2和EP3改成EP4和EP5，因此，MSC的端點處理函數都要修改成對應的端點。

```
void MSC_Read(void)
{
    uint32_t u32Len;

    if (USBD_GET_EP_BUF_ADDR(EP4) == g_u32BulkBuf1)
        USBD_SET_EP_BUF_ADDR(EP4, g_u32BulkBuf0);
    else
        USBD_SET_EP_BUF_ADDR(EP4, g_u32BulkBuf1);

    /* Trigger to send out the data packet */
    USBD_SET_PAYLOAD_LEN(EP4, g_u8Size);

    g_u32Length -= g_u8Size;
    g_u32BytesInStorageBuf -= g_u8Size;

    if (g_u32Length) {
        if (g_u32BytesInStorageBuf) {
            /* Prepare next data packet */
            g_u8Size = EP4_MAX_PKT_SIZE;
            if (g_u8Size > g_u32Length)
                g_u8Size = g_u32Length;

            if (USBD_GET_EP_BUF_ADDR(EP4) == g_u32BulkBuf1)
                USBD_MemCopy((uint8_t *)((uint32_t)USBD_BUF_BASE +
g_u32BulkBuf0), (uint8_t *)g_u32Address, g_u8Size);
```

```
            else
                USBD_MemCopy((uint8_t *)((uint32_t)USBD_BUF_BASE +
g_u32BulkBuf1), (uint8_t *)g_u32Address, g_u8Size);
            g_u32Address += g_u8Size;
        } else {
            u32Len = g_u32Length;
            if (u32Len > STORAGE_BUFFER_SIZE)
                u32Len = STORAGE_BUFFER_SIZE;

            MSC_ReadMedia(g_u32LbaAddress, u32Len, (uint8_t
*)STORAGE_DATA_BUF);
            g_u32BytesInStorageBuf = u32Len;
            g_u32LbaAddress += u32Len;
            g_u32Address = STORAGE_DATA_BUF;

            /* Prepare next data packet */
            g_u8Size = EP4_MAX_PKT_SIZE;
            if (g_u8Size > g_u32Length)
                g_u8Size = g_u32Length;

            if (USBD_GET_EP_BUF_ADDR(EP4) == g_u32BulkBuf1)
                USBD_MemCopy((uint8_t *)((uint32_t)USBD_BUF_BASE +
g_u32BulkBuf0), (uint8_t *)g_u32Address, g_u8Size);
            else
                USBD_MemCopy((uint8_t *)((uint32_t)USBD_BUF_BASE +
g_u32BulkBuf1), (uint8_t *)g_u32Address, g_u8Size);
            g_u32Address += g_u8Size;
        }
    }
}
```

3.2.2 VCOM + HID Transfer

從BSP\SampleCode\StdDriver分別取得VCOM和HID Transfer這兩個獨立功能的範例程式，用VCOM範例程式為基底來與HID Transfer範例合併。使用者必須更動descriptor.c、vcom_serial.c及vcom_serial.h，開發步驟如下：

步驟 1：端點配置及端點初始化

請參考表 2 非多重介面功能複合裝置的介面及端點配置，在USBD_VCOM_SerialEmulator\ vcom_serial.h新增HID Transfer所要使用的端點配置，程式如下：

```
/* 端點最大封包長度 */
#define EP0_MAX_PKT_SIZE    64
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    64
#define EP3_MAX_PKT_SIZE    64
#define EP4_MAX_PKT_SIZE    8
#define EP5_MAX_PKT_SIZE    64
#define EP6_MAX_PKT_SIZE    64

/* 各個端點在USB Buffer的偏移值 */
#define SETUP_BUF_BASE  0
#define SETUP_BUF_LEN   8
#define EP0_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP0_BUF_LEN     EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP1_BUF_LEN     EP1_MAX_PKT_SIZE
#define EP2_BUF_BASE    (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN     EP2_MAX_PKT_SIZE
#define EP3_BUF_BASE    (EP2_BUF_BASE + EP2_BUF_LEN)
#define EP3_BUF_LEN     EP3_MAX_PKT_SIZE
#define EP4_BUF_BASE    (EP3_BUF_BASE + EP3_BUF_LEN)
#define EP4_BUF_LEN     EP4_MAX_PKT_SIZE
#define EP5_BUF_BASE    (EP4_BUF_BASE + EP4_BUF_LEN)
#define EP5_BUF_LEN     EP5_MAX_PKT_SIZE
#define EP6_BUF_BASE    (EP5_BUF_BASE + EP5_BUF_LEN)
#define EP6_BUF_LEN     EP6_MAX_PKT_SIZE

/* 端點地址 */
#define BULK_IN_EP_NUM        0x01
#define BULK_OUT_EP_NUM       0x02
#define INT_IN_EP_NUM         0x03
#define INT_IN_EP_NUM_1       0x04
#define INT_OUT_EP_NUM_1      0x05
```

端點配置完之後，在USBD_VCOM_SerialEmulator\vcom_serial.c中的VCOM_Init()裡面新增HID的端點初始化。

```
void VCOM_Init(void)
{
```

```
    /* Init setup packet buffer */
    /* Buffer for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;

    /****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

    /****************************************************/
    /* EP2 ==> Bulk IN endpoint, address 1 */
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | BULK_IN_EP_NUM);
    /* Buffer offset for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

    /* EP3 ==> Bulk Out endpoint, address 2 */
    USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_OUT | BULK_OUT_EP_NUM);
    /* Buffer offset for EP3 */
    USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);
    /* trigger receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);

    /* EP4 ==> Interrupt IN endpoint, address 3 */
    USBD_CONFIG_EP(EP4, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
    /* Buffer offset for EP4 ->  */
    USBD_SET_EP_BUF_ADDR(EP4, EP4_BUF_BASE);

    /****************************************************/
    /* EP5 ==> Interrupt IN endpoint, address 4 */
    USBD_CONFIG_EP(EP5, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
    /* Buffer range for EP5 */
    USBD_SET_EP_BUF_ADDR(EP5, EP5_BUF_BASE);
```

```
    /* EP6 ==> Interrupt OUT endpoint, address 5 */
    USBD_CONFIG_EP(EP6, USBD_CFG_EPMODE_OUT | INT_OUT_EP_NUM);
    /* Buffer range for EP6 */
    USBD_SET_EP_BUF_ADDR(EP6, EP6_BUF_BASE);
    /* trigger to receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP6, EP6_MAX_PKT_SIZE);
}
```

步驟 2：定義描述元

因為此範例屬於多重介面的複合裝置，所以需要額外使用介面關聯描述元(IAD)來定義多重介面功能，因此在USBD_VCOM_SerialEmulator\descriptor.c的設備描述元中的bDeviceClass、bDeviceSubClass及bDeviceProtocol分別修改為：0xEF, 0x02, 0x01。

```
/* USB設備描述元 */
uint8_t gu8DeviceDescriptor[] = {
    LEN_DEVICE,         /* bLength */
    DESC_DEVICE,        /* bDescriptorType */
    0x00, 0x02,         /* bcdUSB */
    0xEF,               /* bDeviceClass: IAD*/
    0x02,               /* bDeviceSubClass */
    0x01,               /* bDeviceProtocol */
    EP0_MAX_PKT_SIZE,   /* bMaxPacketSize0 */
    /* idVendor */
    USBD_VID & 0x00FF,
    (USBD_VID & 0xFF00) >> 8,
    /* idProduct */
    USBD_PID & 0x00FF,
    (USBD_PID & 0xFF00) >> 8,
    0x00, 0x03,     /* bcdDevice */
    0x01,           /* iManufacture */
    0x02,           /* iProduct */
    0x00,           /* iSerialNumber - no serial */
    0x01            /* bNumConfigurations */
}
```

額外使用IAD來定義多重介面功能，除了修改設備描述元之外，另外還要在組態描述元中加上IAD描述元來將某些介面關聯成同一個功能的介面。首先在兩個要合併的介面前面加上IAD描述元，要關聯的兩個介面描述元需要緊接在IAD描述元之後，其中IAD描述元中的bFirstInterface表示與此功能關聯的第一個介面的介面編

號，bInterfaceCount則代表連續幾個介面關聯組合成一個功能，本範例的VCOM是把兩個介面關聯在一起，所以bInterfaceCount為2。

在USBD_VCOM_SerialEmulator\descriptor.c新增HID類別特定的類別描述元及報告描述元，並且在組態描述元裡面新增HID裝置的介面描述元及其使用的端點描述元。在這裡需要注意wTotalLength、bNumInterfaces及bInterfaceNumber。其中，WTotalLength則為組態描述元的總長度。另外，這個範例VCOM使用兩個介面，而HID Transfer使用一個介面，因此，bNumInterfaces值為3，其中在介面描述元的HID介面bInterfaceNumber為2。

```
/* USB 組態描述元，它包含介面、HID及端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
    LEN_CONFIG,         /* bLength              */
    DESC_CONFIG,        /* bDescriptorType      */

    /* wTotalLength = LEN_CONFIG+LEN_INTERFACE*2+LEN_HID+LEN_ENDPOINT*5 */
    0x6B, 0x00,         /* wTotalLength         */
    0x03,               /* bNumInterfaces       */
    0x01,               /* bConfigurationValue  */
    0x00,               /* iConfiguration       */
    0xC0,               /* bmAttributes         */
    0x32,               /* MaxPower             */

/* 介面關聯描述元(IAD) */
    0x08,               // bLength: Interface Descriptor size
    0x0B,               // bDescriptorType: IAD
    0x00,               // bFirstInterface
    0x02,               // bInterfaceCount
    0x02,               // bFunctionClass: CDC
    0x02,               // bFunctionSubClass
    0x01,               // bFunctionProtocol
    0x02,               // iFunction

/* 介面0 : VCOM Control介面描述元*/
    LEN_INTERFACE,  /* bLength              */
    DESC_INTERFACE, /* bDescriptorType      */
    0x00,           /* bInterfaceNumber     */
    0x00,           /* bAlternateSetting    */
    0x01,           /* bNumEndpoints        */
    0x02,           /* bInterfaceClass      */
```

```
    0x02,           /* bInterfaceSubClass   */
    0x01,           /* bInterfaceProtocol   */
    0x00,           /* iInterface           */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x00,           /* Header functional descriptor subtype */
    0x10, 0x01,     /* Communication device compliant to the communication
spec. ver. 1.10 */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x01,           /* Call management functional descriptor */
    0x00,           /* BIT0: Whether device handle call management itself. */
                    /* BIT1: Whether device can send/receive call management
information over a Data Class Interface 0 */
    0x01,           /* Interface number of data class interface optionally
used for call management */

    /* Communication Class Specified INTERFACE descriptor */
    0x04,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x02,           /* Abstract control management functional descriptor
subtype */
    0x00,           /* bmCapabilities       */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* bLength              */
    0x24,           /* bDescriptorType: CS_INTERFACE descriptor type */
    0x06,           /* bDescriptorSubType   */
    0x00,           /* bMasterInterface     */
    0x01,           /* bSlaveInterface0     */

/* 端點4描述元: 中斷IN，地址為INT_IN_EP_NUM */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_INPUT | INT_IN_EP_NUM),     /* bEndpointAddress */
    EP_INT,                         /* bmAttributes     */
```

```
    EP4_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x01,                           /* bInterval        */

/* 介面1 : VCOM Data介面描述元 */
    LEN_INTERFACE,                  /* bLength              */
    DESC_INTERFACE,                 /* bDescriptorType      */
    0x01,                           /* bInterfaceNumber     */
    0x00,                           /* bAlternateSetting    */
    0x02,                           /* bNumEndpoints        */
    0x0A,                           /* bInterfaceClass      */
    0x00,                           /* bInterfaceSubClass   */
    0x00,                           /* bInterfaceProtocol   */
    0x00,                           /* iInterface           */

/* 端點2描述元: BULK IN，地址為BULK_IN_EP_NUM */
    /* ENDPOINT descriptor */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_INPUT | BULK_IN_EP_NUM),    /* bEndpointAddress */
    EP_BULK,                        /* bmAttributes     */
    EP2_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x00,                           /* bInterval        */

/* 端點3描述元: BULK OUT，地址為BULK_OUT_EP_NUM */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_OUTPUT | BULK_OUT_EP_NUM),  /* bEndpointAddress */
    EP_BULK,                        /* bmAttributes     */
    EP3_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x00,                           /* bInterval        */

/* 介面2 : HID Transfer介面描述元 */
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
    0x02,           /* bInterfaceNumber */
    0x00,           /* bAlternateSetting */
    0x02,           /* bNumEndpoints */
    0x03,           /* bInterfaceClass */
    0x00,           /* bInterfaceSubClass */
```

```
    0x00,           /* bInterfaceProtocol */
    0x00,           /* iInterface */

/* HID描述元 */
    LEN_HID,        /* Size of this descriptor in UINT8s. */
    DESC_HID,       /* HID descriptor type. */
    0x10, 0x01,     /* HID Class Spec. release number. */
    0x00,           /* H/W target country. */
    0x01,           /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,   /* Descriptor type. */
    /* Total length of report descriptor. */
    sizeof(HID_DeviceReportDescriptor) & 0x00FF,
    (sizeof(HID_DeviceReportDescriptor) & 0xFF00) >> 8,

/* 端點5描述元: 中斷IN，地址為INT_IN_EP_NUM_1 */
    LEN_ENDPOINT,                       /* bLength */
    DESC_ENDPOINT,                      /* bDescriptorType */
    (INT_IN_EP_NUM_1 | EP_INPUT),       /* bEndpointAddress */
    EP_INT,                             /* bmAttributes */
    /* wMaxPacketSize */
    EP5_MAX_PKT_SIZE & 0x00FF,
    (EP5_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,        /* bInterval */

/* 端點6描述元: 中斷OUT，地址為INT_OUT_EP_NUM_1 */
    LEN_ENDPOINT,                       /* bLength */
    DESC_ENDPOINT,                      /* bDescriptorType */
    (INT_OUT_EP_NUM_1 | EP_OUTPUT),     /* bEndpointAddress */
    EP_INT,                             /* bmAttributes */
    /* wMaxPacketSize */
    EP6_MAX_PKT_SIZE & 0x00FF,
    (EP6_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,        /* bInterval */
};

/* HID Transfer Report描述元 */
uint8_t HID_DeviceReportDescriptor[] =
{
    0x06, 0x00, 0xFF,   // Usage Page = 0xFF00 (Vendor Defined Page 1)
```

```
    0x09, 0x01,         // Usage (Vendor Usage 1)
    0xA1, 0x01,         // Collection (Application)
    0x19, 0x01,         // Usage Minimum
    0x29, 0x40,         // Usage Maximum //64 input usages total (0x01 to
0x40)
    0x15, 0x00,         // Logical Minimum (data bytes in the report may have
minimum value = 0x00)
    0x26, 0xFF, 0x00,   // Logical Maximum (data bytes in the report may have
maximum value = 0x00FF = unsigned 255)
    0x75, 0x08,         // Report Size: 8-bit field size
    0x95, 0x40,         // Report Count: Make sixty-four 8-bit fields (the
next time the parser hits
                        // an "Input", "Output", or "Feature" item)
    0x81, 0x00,         // Input (Data, Array, Abs): Instantiates input packet
fields based on the
                        // above report size, count, logical min/max, and
usage.
    0x19, 0x01,         // Usage Minimum
    0x29, 0x40,         // Usage Maximum //64 output usages total (0x01 to
0x40)
    0x91, 0x00,         // Output (Data, Array, Abs): Instantiates output
packet fields. Uses same
                        // report size and count as "Input" fields, since
nothing new/different was
                        // specified to the parser since the "Input" item.
    0xC0                // End Collection
};
```

因為此複合裝置的介面2為HID功能，所以在實作時新增與HID特定的類別描述元相關的結構。USBD_VCOM_SerialEmulator\descriptor.c修改程式如下所示。

```
uint8_t *gu8UsbHidReport[3] = {
    NULL,
    NULL,
    HID_DeviceReportDescriptor,            /* HID transfer報告描述元 */
};

uint32_t gu32UsbHidReportLen[3] = {
    0,
    0,
    sizeof(HID_DeviceReportDescriptor),    /* HID transfer報告描述元的長度 */
};
```

```
uint32_t gu32ConfigHidDescIdx[3] = {
    0,
    0,
/* HID transfer描述元在組態描述元內的位置 */
    (sizeof(gu8ConfigDescriptor) - LEN_HID - (2*LEN_ENDPOINT)),
};
```

步驟 3：定義類別的特定請求

新增HID設備的六個類別請求，直接從USBD_HID_Transfer\hid_transfer.h把HID的六個請求定義複製到USBD_VCOM_SerialEmulator\vcom_serial.h。

```
/*!<Define CDC Class Specific Request */
#define SET_LINE_CODE           0x20
#define GET_LINE_CODE           0x21
#define SET_CONTROL_LINE_STATE  0x22

/*!<Define HID Class Specific Request */
#define GET_REPORT          0x01
#define GET_IDLE            0x02
#define GET_PROTOCOL        0x03
#define SET_REPORT          0x09
#define SET_IDLE            0x0A
#define SET_PROTOCOL        0x0B

/*!<USB HID Interface Class protocol */
#define HID_NONE            0x00
#define HID_KEYBOARD        0x01
#define HID_MOUSE           0x02

/*!<USB HID Class Report Type */
#define HID_RPT_TYPE_INPUT      0x01
#define HID_RPT_TYPE_OUTPUT     0x02
#define HID_RPT_TYPE_FEATURE    0x03
```

VID和PID決定USB主機要安裝哪個驅動程式，USBD_VCOM_SerialEmulator\vcom_serial.h請參考表 4修改成此複合裝置所對應的VID/PID。

```
#define USBD_VID        0x0416
#define USBD_PID        0xDC00
```

本範例的驅動程式只要修改USBD_VCOM_SerialEmulator\Windows Driver\NuvotonCDC.inf中的PID即可。

```
[Devices.NT]
%DESCRIPTION%= NuvotonCDC_DEV, USB\VID_0416&PID_DC00&MI_00
[Devices.NTamd64]
%DESCRIPTION%= NuvotonCDC_DEV, USB\VID_0416&PID_DC00&MI_00
```

步驟 4：回應類別的特定請求

USBD_VCOM_SerialEmulator\vcom_serial.c加入回應HID設備所需要的六個類別請求。使用者可以從USBD_HID_Transfer\hid_transfer.c複製回應HID的六個請求的程式碼，再依照步驟1的端點配置修改為相對應的地址及傳輸類型。

```
void VCOM_ClassRequest(void)
{
    uint8_t buf[8];
    USBD_GetSetupPacket(buf);

    if (buf[0] & 0x80) { /* request data transfer direction */
        // Device to host
        switch (buf[1]) {
        case GET_LINE_CODE: {
            if (buf[4] == 0) { /* VCOM-1 */
                USBD_MemCopy((uint8_t *)(USBD_BUF_BASE +
USBD_GET_EP_BUF_ADDR(EP0)), (uint8_t *)&gLineCoding, 7);
            }
            /* Data stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 7);
            /* Status stage */
            USBD_PrepareCtrlOut(0,0);
            break;
        }
        case GET_REPORT:
        case GET_IDLE:
        case GET_PROTOCOL:
        default: {
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
        }
    } else {
```

```
        // Host to device
        switch (buf[1]) {
        case SET_CONTROL_LINE_STATE: {
            if (buf[4] == 0) { /* VCOM-1 */
                gCtrlSignal = buf[3];
                gCtrlSignal = (gCtrlSignal << 8) | buf[2];
            }
            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;
        }
        case SET_LINE_CODE: {
            if (buf[4] == 0) /* VCOM-1 */
                USBD_PrepareCtrlOut((uint8_t *)&gLineCoding, 7);
            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;
        }
        case SET_REPORT: {
            if (buf[3] == 3) {
            /* Request Type = Feature */
            USBD_SET_DATA1(EP1);
            USBD_SET_PAYLOAD_LEN(EP1, 0);
              }
            break;
        }
        case SET_IDLE: {
             /* Status stage */
             USBD_SET_DATA1(EP0);
             USBD_SET_PAYLOAD_LEN(EP0, 0);
             break;
        }
        case SET_PROTOCOL:
        default: {
             // Stall
             /* Setup error, stall the device */
             USBD_SetStall(0);
```

```
                break;
            }
        }
    }
}
```

步驟 5：處理每個端點發生的事件

USBD_VCOM_SerialEmulator\vcom_serial.c中的USBD_IRQHandler()得知哪個端點發生中斷，並且執行該端點的Callback function。使用者在USBD_IRQHandler()中新增HID端點5和端點6的Callback function。

```
void USBD_IRQHandler(void)
{
    uint32_t u32IntSts = USBD_GET_INT_FLAG();
    uint32_t u32State = USBD_GET_BUS_STATE();

//-----------------------------------------------------------------
    if (u32IntSts & USBD_INTSTS_FLDET) {
        // Floating detect
        USBD_CLR_INT_FLAG(USBD_INTSTS_FLDET);

        if (USBD_IS_ATTACHED()) {
            /* USB Plug In */
            USBD_ENABLE_USB();
        } else {
            /* USB Un-plug */
            USBD_DISABLE_USB();
        }
    }
//-----------------------------------------------------------------
    if (u32IntSts & USBD_INTSTS_BUS) {
        /* Clear event flag */
        USBD_CLR_INT_FLAG(USBD_INTSTS_BUS);
        if (u32State & USBD_STATE_USBRST) {
            /* Bus reset */
            USBD_ENABLE_USB();
            USBD_SwReset();
        }
        if (u32State & USBD_STATE_SUSPEND) {
            /* Enable USB but disable PHY */
```

```
            USBD_DISABLE_PHY();
        }
        if (u32State & USBD_STATE_RESUME) {
            /* Enable USB and enable PHY */
            USBD_ENABLE_USB();
        }
    }
//-------------------------------------------------------------------
    if (u32IntSts & USBD_INTSTS_USB) {
        extern uint8_t g_usbd_SetupPacket[];
        // USB event
        if (u32IntSts & USBD_INTSTS_SETUP) {
            // Setup packet
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_SETUP);
            /* Clear the data IN/OUT ready flag of control end-points */
            USBD_STOP_TRANSACTION(EP0);
            USBD_STOP_TRANSACTION(EP1);
            USBD_ProcessSetupPacket();
        }
        // EP events
        if (u32IntSts & USBD_INTSTS_EP0) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP0);
            // control IN
            USBD_CtrlIn();
        }
        if (u32IntSts & USBD_INTSTS_EP1) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP1);
            // control OUT
            USBD_CtrlOut();
            // In ACK of SET_LINE_CODE
            if(g_usbd_SetupPacket[1] == SET_LINE_CODE)
            {
                if(g_usbd_SetupPacket[4] == 0)  /* VCOM-1 */
                    VCOM_LineCoding(0); /* Apply UART settings */
            }
        }
```

```
        if (u32IntSts & USBD_INTSTS_EP2) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP2);
            // Bulk IN
            EP2_Handler();
        }
        if (u32IntSts & USBD_INTSTS_EP3) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP3);
            // Bulk Out
            EP3_Handler();
        }
        if (u32IntSts & USBD_INTSTS_EP4) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP4);
        }
        if (u32IntSts & USBD_INTSTS_EP5) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP5);
            // Interrupt IN
            EP5_Handler();
        }
        if (u32IntSts & USBD_INTSTS_EP6) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP6);
            // Interrupt OUT
            EP6_Handler();
        }
        if (u32IntSts & USBD_INTSTS_EP7) {
            /* Clear event flag */
            USBD_CLR_INT_FLAG(USBD_INTSTS_EP7);
        }
    }
    /* clear unknown event */
    USBD_CLR_INT_FLAG(u32IntSts);
}
void EP2_Handler(void)
{
    gu32TxSize = 0;
```

```
}
void EP3_Handler(void)
{
    /* Bulk OUT */
    gu32RxSize = USBD_GET_PAYLOAD_LEN(EP3);
    gpu8RxBuf = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP3));
    /* Set a flag to indicate bulk out ready */
    gi8BulkOutReady = 1;
}
void EP5_Handler(void)  /* Interrupt IN handler */
{
    HID_SetInReport();
}
void EP6_Handler(void)  /* Interrupt OUT handler */
{
    uint8_t *ptr;
    /* Interrupt OUT */
    ptr = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP6));
    HID_GetOutReport(ptr, USBD_GET_PAYLOAD_LEN(EP6));
    USBD_SET_PAYLOAD_LEN(EP6, EP6_MAX_PKT_SIZE);
}
```

最後，把USBD_HID_Transfer\hid_transfer.c裡面HID相關的端點處理程式複製至USBD_VCOM_SerialEmulator\vcom_serial.c，這樣就完成VCOM+HID Transfer的複合裝置。這裡要特別注意每個裝置所對應的端點處理程式，程式必須根據其對應的端點來做修改，否則會無法成功辨識此裝置，造成裝置無法使用。下面的程式碼以HID_SetInReport()來簡單說明。這個範例HID Transfer端點是使用EP5和EP6，而HID_SetInReport()是透過EP5來傳送資料，因此要把原本的這段程式使用的EP2改成EP5。

```
void HID_SetInReport(void)
{
    uint32_t u32StartPage;
    uint32_t u32TotalPages;
    uint32_t u32PageCnt;
    uint8_t *ptr;
    uint8_t u8Cmd;
    u8Cmd        = gCmd.u8Cmd;
    u32StartPage = gCmd.u32Arg1;
    u32TotalPages= gCmd.u32Arg2;
```

```
    u32PageCnt   = gCmd.u32Signature;

    /* Check if it is in data phase of read command */
    if(u8Cmd == HID_CMD_READ)
    {
        /* Process the data phase of read command */
        if((u32PageCnt >= u32TotalPages) && (g_u32BytesInPageBuf == 0))
        {
            /* The data transfer is complete. */
            u8Cmd = HID_CMD_NONE;
            printf("Read command complete!\n");
        }
        else
        {
            if(g_u32BytesInPageBuf == 0)
            {
            /* The previous page has sent out. Read new page to page buffer */
            memcpy(g_u8PageBuff, g_u8TestPages + u32PageCnt * PAGE_SIZE,
sizeof(g_u8PageBuff));
            g_u32BytesInPageBuf = PAGE_SIZE;
            /* Update the page counter */
            u32PageCnt++;
            }

            /* Prepare the data for next HID IN transfer */
            ptr = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP5));
            USBD_MemCopy(ptr, (void *)&g_u8PageBuff[PAGE_SIZE -
g_u32BytesInPageBuf], EP5_MAX_PKT_SIZE);
            USBD_SET_PAYLOAD_LEN(EP5, EP5_MAX_PKT_SIZE);
            g_u32BytesInPageBuf -= EP5_MAX_PKT_SIZE;
        }
    }
    gCmd.u8Cmd        = u8Cmd;
    gCmd.u32Signature = u32PageCnt;
}
```

3.3 應用範例

各種應用的類別實作包括類別請求、端點資料處理，接下來的章節會介紹各個應用範例要修改哪些相對應的應用層檔案，以適應各種不同的USB類別應用。

3.3.1 HID Transfer + MSC

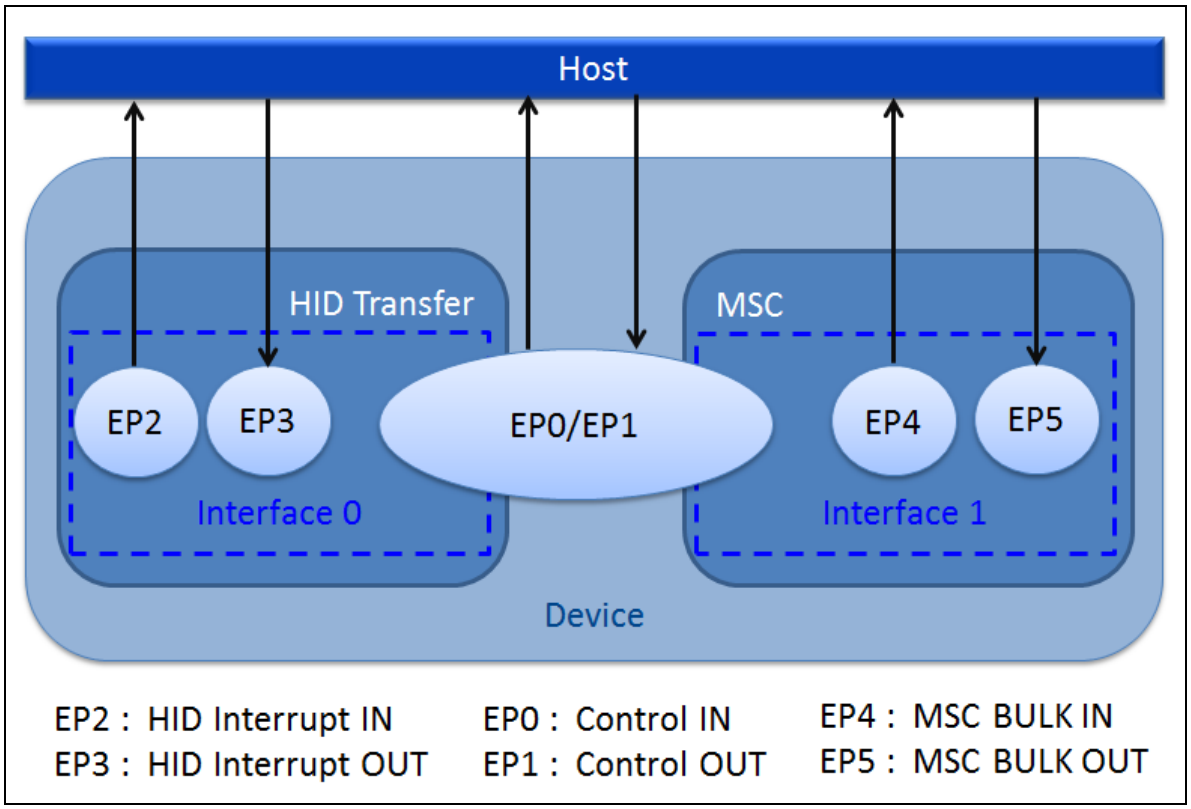


圖 7 HID Transfer+MSC複合裝置的架構

這個複合裝置範例是結合HID Transfer和USB MSC，圖 7說明HID Transfer和MSC複合裝置範例的架構。首先可以先從BSP\SampleCode\StdDriver分別取得HID Transfer和MSC這兩個獨立功能的範例程式，使用者可以藉由使用HID Transfer的範例程式並結合MSC的範例程式來實作出新的HID Transfer和USB MSC的複合裝置範例。開發步驟如下：

步驟1：根據裝置的類別來配置端點和其對應的IN/OUT Buffer，此範例配置六個端點EP0~EP5。EP0_BUF_BASE~EP5_BUF_BASE分別為各個端點可以使用的Buffer起始位址。EP0被配置為Control IN，EP1被配置為Control OUT，EP0和EP1的最大封包長度皆為8Bytes。EP2被配置為Interrupt IN，EP3被配置為Interrupt OUT，EP4被配置為BULK IN，EP5被配置為BULK OUT，EP2~EP5的最大封包長度皆為64Bytes。下面的程式碼為端點配置相關設定。

```
/* 端點最大封包長度 */
#define EP0_MAX_PKT_SIZE    8
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    64
```

```
#define EP3_MAX_PKT_SIZE    64
#define EP4_MAX_PKT_SIZE    64
#define EP5_MAX_PKT_SIZE    64

/* 各個端點在USB Buffer的偏移值 */
#define SETUP_BUF_BASE  0
#define SETUP_BUF_LEN   8
#define EP0_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP0_BUF_LEN     EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP1_BUF_LEN     EP1_MAX_PKT_SIZE
#define EP2_BUF_BASE    (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN     EP2_MAX_PKT_SIZE
#define EP3_BUF_BASE    (EP2_BUF_BASE + EP2_BUF_LEN)
#define EP3_BUF_LEN     EP3_MAX_PKT_SIZE
#define EP4_BUF_BASE    (EP3_BUF_BASE + EP3_BUF_LEN)
#define EP4_BUF_LEN     EP4_MAX_PKT_SIZE
#define EP5_BUF_BASE    (EP4_BUF_BASE + EP4_BUF_LEN)
#define EP5_BUF_LEN     EP5_MAX_PKT_SIZE

void HID_Init(void)
{
    /* Init setup packet buffer */
    /* Buffer range for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;

    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

    /*****************************************************/
    /* EP2 ==> Interrupt IN endpoint, address 1 */
```

```
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
    /* Buffer range for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

    /* EP3 ==> Interrupt OUT endpoint, address 2 */
    USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_OUT | INT_OUT_EP_NUM);
    /* Buffer range for EP3 */
    USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);
    /* trigger to receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);

    /*****************************************************/
    /* EP4 ==> Bulk IN endpoint, address 3 */
    USBD_CONFIG_EP(EP4, USBD_CFG_EPMODE_IN | BULK_IN_EP_NUM);
    /* Buffer range for EP4 */
    USBD_SET_EP_BUF_ADDR(EP4, EP4_BUF_BASE);

    /* EP5 ==> Bulk Out endpoint, address 4 */
    USBD_CONFIG_EP(EP5, USBD_CFG_EPMODE_OUT | BULK_OUT_EP_NUM);
    /* Buffer range for EP5 */
    USBD_SET_EP_BUF_ADDR(EP5, EP5_BUF_BASE);
    /* trigger to receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP5, EP5_MAX_PKT_SIZE);
}
```

步驟2：設備、配置、介面、端點和字串描述元都包含在"descriptor.c"中，使用者在此檔案中加入HID Transfer和MSC這兩個類別的各種描述元。

```
/* USB設備描述元， VID和PID決定USB主機要安裝哪個驅動程式 */
uint8_t gu8DeviceDescriptor[] = {
    LEN_DEVICE,     /* bLength */
    DESC_DEVICE,    /* bDescriptorType */
    0x10, 0x01,     /* bcdUSB */
    0x00,           /* bDeviceClass */
    0x00,           /* bDeviceSubClass */
    0x00,           /* bDeviceProtocol */
    EP0_MAX_PKT_SIZE,   /* bMaxPacketSize0 */
    /* idVendor */
    USBD_VID & 0x00FF,
    (USBD_VID & 0xFF00) >> 8,
```

```
    /* idProduct */
    USBD_PID & 0x00FF,
    (USBD_PID & 0xFF00) >> 8,
    0x00, 0x00,     /* bcdDevice */
    0x01,           /* iManufacture */
    0x02,           /* iProduct */
    0x02,           /* iSerialNumber */
    0x01            /* bNumConfigurations */
};
/* USB 組態描述元，它包含介面、HID及端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
/* 組態描述元 */
    LEN_CONFIG,     /* bLength */
    DESC_CONFIG,    /* bDescriptorType */
    /* wTotalLength */
    0x40,
    0x00,
    0x02,           /* bNumInterfaces */
    0x01,           /* bConfigurationValue */
    0x00,           /* iConfiguration */
    0x80 | (USBD_SELF_POWERED << 6) | (USBD_REMOTE_WAKEUP << 5),/*
bmAttributes */
    USBD_MAX_POWER, /* MaxPower */

/* 介面0 : HID Transfer介面描述元*/
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
    0x00,           /* bInterfaceNumber */
    0x00,           /* bAlternateSetting */
    0x02,           /* bNumEndpoints */
    0x03,           /* bInterfaceClass */
    0x00,           /* bInterfaceSubClass */
    0x00,           /* bInterfaceProtocol */
    0x00,           /* iInterface */

/* HID描述元 */
    LEN_HID,        /* Size of this descriptor in UINT8s. */
    DESC_HID,       /* HID descriptor type. */
    0x10, 0x01,     /* HID Class Spec. release number. */
```

```
    0x00,           /* H/W target country. */
    0x01,           /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,   /* Descriptor type. */
    /* Total length of report descriptor. */
    sizeof(HID_DeviceReportDescriptor) & 0x00FF,
    (sizeof(HID_DeviceReportDescriptor) & 0xFF00) >> 8,

/* 端點2描述元: 中斷IN，地址為INT_IN_EP_NUM */
    LEN_ENDPOINT,                               /* bLength */
    DESC_ENDPOINT,                              /* bDescriptorType */
    (INT_IN_EP_NUM | EP_INPUT),                 /* bEndpointAddress */
    EP_INT,                                     /* bmAttributes */
    /* wMaxPacketSize */
    EP2_MAX_PKT_SIZE & 0x00FF,
    (EP2_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,                /* bInterval */

/* 端點3描述元: 中斷OUT，地址為INT_OUT_EP_NUM */
    LEN_ENDPOINT,                               /* bLength */
    DESC_ENDPOINT,                              /* bDescriptorType */
    (INT_OUT_EP_NUM | EP_OUTPUT),               /* bEndpointAddress */
    EP_INT,                                     /* bmAttributes */
    /* wMaxPacketSize */
    EP3_MAX_PKT_SIZE & 0x00FF,
    (EP3_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,                /* bInterval */

/* 介面1：MSC 介面描述元 */
    LEN_INTERFACE,                              // bLength
    DESC_INTERFACE,                             // bDescriptorType
    0x01,                                       // bInterfaceNumber
    0x00,                                       // bAlternateSetting
    0x02,                                       // bNumEndpoints
    0x08,                                       // bInterfaceClass
    0x06,                                       // bInterfaceSubClass
    0x50,                                       // bInterfaceProtocol
    0x00,                                       // iInterface

/* 端點4描述元: BULK IN，地址為BULK_IN_EP_NUM */
```

```
    LEN_ENDPOINT,                                   // bLength
    DESC_ENDPOINT,                                  // bDescriptorType
    (EP_INPUT | BULK_IN_EP_NUM),                    // bEndpointAddress
    EP_BULK,                                        // bmAttributes
    EP4_MAX_PKT_SIZE, 0x00,                         // wMaxPacketSize
    0x00,                                           // bInterval

/* 端點5描述元: BULK OUT，地址為BULK_OUT_EP_NUM */
    LEN_ENDPOINT,                                   // bLength
    DESC_ENDPOINT,                                  // bDescriptorType
    (EP_OUTPUT | BULK_OUT_EP_NUM),                  // bEndpointAddress
    EP_BULK,                                        // bmAttributes
    EP5_MAX_PKT_SIZE, 0x00,                         // wMaxPacketSize
    0x00                                            // bInterval
};

/* HID Transfer Report描述元 */
uint8_t HID_DeviceReportDescriptor[] =
{
    0x06, 0x00, 0xFF,   // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,         // Usage (Vendor Usage 1)
    0xA1, 0x01,         // Collection (Application)
    0x19, 0x01,         // Usage Minimum
    0x29, 0x40,         // Usage Maximum
    0x15, 0x00,         // Logical Minimum
    0x26, 0xFF, 0x00,   // Logical Maximum
    0x75, 0x08,         // Report Size: 8-bit field size
    0x95, 0x40,         // Report Count
    // an "Input", "Output", or "Feature" item)
    0x81, 0x00,         // Input (Data, Array, Abs)
   // above report size, count, logical min/max, and usage.
    0x19, 0x01,         // Usage Minimum
    0x29, 0x40,         // Usage Maximum
    0x91, 0x00,         // Output (Data, Array, Abs)
    0xC0                // End Collection
};

/* 字串描述元 */
uint8_t gu8StringLang[4] = {
```

```
    4,                                          /* bLength */
    DESC_STRING,                                /* bDescriptorType */
    0x09, 0x04
};

/* 字串描述元 */
uint8_t gu8VendorStringDesc[] = {
    16,
    DESC_STRING,
    'N', 0, 'u', 0, 'v', 0, 'o', 0, 't', 0, 'o', 0, 'n', 0
};

/* 字串描述元 */
uint8_t gu8ProductStringDesc[] = {
    22,                                         /* bLength          */
    DESC_STRING,                                /* bDescriptorType  */
    'U', 0, 'S', 0, 'B', 0, ' ', 0, 'D', 0, 'e', 0, 'v', 0, 'i', 0, 'c', 0,
'e', 0
};

/* 因為此範例的HID為介面0，因此HID相關結構的Index皆為0 */
uint8_t *gu8UsbHidReport[3] = {
    HID_DeviceReportDescriptor,             /* HID transfer報告描述元 */
    NULL,
    NULL,
};

uint32_t gu32UsbHidReportLen[3] = {
    sizeof(HID_DeviceReportDescriptor),     /* HID transfer報告描述元的長度 */
    0,
    0,
};

uint32_t gu32ConfigHidDescIdx[3] = {
    (LEN_CONFIG+LEN_INTERFACE),     /* HID transfer描述元在組態描述元內的位置*/
    0,
    0,
};
```

步驟3：hid_transfer_And_MSC.c執行HID/MSC類別請求和端點資料處理，USB在HID設備中定義了六個類別請求，MSC定義兩個類別請求，但是並不是所有的命令都必須回應。hid_transfer_And_MSC.c執行HID/MSC類別請求及端點EP2~EP5的Callback如下所示：

```
void HID_ClassRequest(void)
{
    uint8_t buf[8];
    USBD_GetSetupPacket(buf);

    if (buf[0] & 0x80) {                /* 該命令數據傳輸方向是Device -> Host */
        switch (buf[1]) {
        case GET_MAX_LUN: {             /* 介面1: MSC的類別請求 */
             if ((((buf[3]<<8)+buf[2]) == 0) && (((buf[5]<<8)+buf[4]) == 0) &&
(((buf[7]<<8)+buf[6]) == 1))
             {
                M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = 0;
                /* Data stage */
                USBD_SET_DATA1(EP0);
                USBD_SET_PAYLOAD_LEN(EP0, 1);
                /* Status stage */
                USBD_PrepareCtrlOut(0,0);
            }
            else {  /* Invalid Get MaxLun command */
                USBD_SetStall(0);
            }
            break;
        }
        /* 介面0: HID的類別請求 */
        case GET_REPORT:                                        /* 不處理 */
        case GET_IDLE:                                          /* 不處理 */
        case GET_PROTOCOL:                                      /* 不處理 */
        default: {
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
        }
    } else {
        /* Host -> Device */
```

```
        switch (buf[1]) {
        /* 介面0: HID的類別請求 */
        case SET_REPORT: {
            if (buf[3] == 3) {
                /* Request Type = Feature */
                USBD_SET_DATA1(EP1);
                USBD_SET_PAYLOAD_LEN(EP1, 0);
            }
            break;
        }
        /* 介面0: HID的類別請求 */
        case SET_IDLE: {
            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;
        }
        /* 介面1: MSC的類別請求 */
        case BULK_ONLY_MASS_STORAGE_RESET: {
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            USBD_LockEpStall(0);

            /* Clear ready */
            USBD->EP[EP4].CFG |= USBD_CFG_CLRRDY_Msk;
            USBD->EP[EP5].CFG |= USBD_CFG_CLRRDY_Msk;

            /* Prepare to receive the CBW */
            g_u8EP5Ready = 0;
            g_u8BulkState = BULK_CBW;

            USBD_SET_DATA1(EP5);
            USBD_SET_EP_BUF_ADDR(EP5, g_u32BulkBuf0);
            USBD_SET_PAYLOAD_LEN(EP5, EP5_MAX_PKT_SIZE);

            break;
        }
        /* 介面0: HID的類別請求 */
        case SET_PROTOCOL:                                          /* 不處理 */
```

```
        default: {
            // Stall
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
        }
    }
}

void EP2_Handler(void)  /* Interrupt IN handler */
{
    HID_SetInReport();
}

void EP3_Handler(void)  /* Interrupt OUT handler */
{
    uint8_t *ptr;
    /* Interrupt OUT */
    ptr = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP3));
    HID_GetOutReport(ptr, USBD_GET_PAYLOAD_LEN(EP3));
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
}

void EP4_Handler(void)  /* BULK IN handler */
{
    g_u8EP4Ready = 1;
    MSC_AckCmd();
}

void EP5_Handler(void) /* BULK OUT handler */
{
    /* Bulk OUT */
    g_u8EP5Ready = 1;
}
```

執行此範例當裝置列舉成功時，可在電腦的裝置管理員中看到HID Transfer + MSC的複合裝置，不需另外安裝驅動程式，如圖 8所示。HID Transfer可透過PC端的Windows Tool來與HID裝置互相通訊，請參閱Windows Tool；另一個MSC裝置則

在PC端辨識為移動式硬碟，可以直接對它做檔案的讀寫。

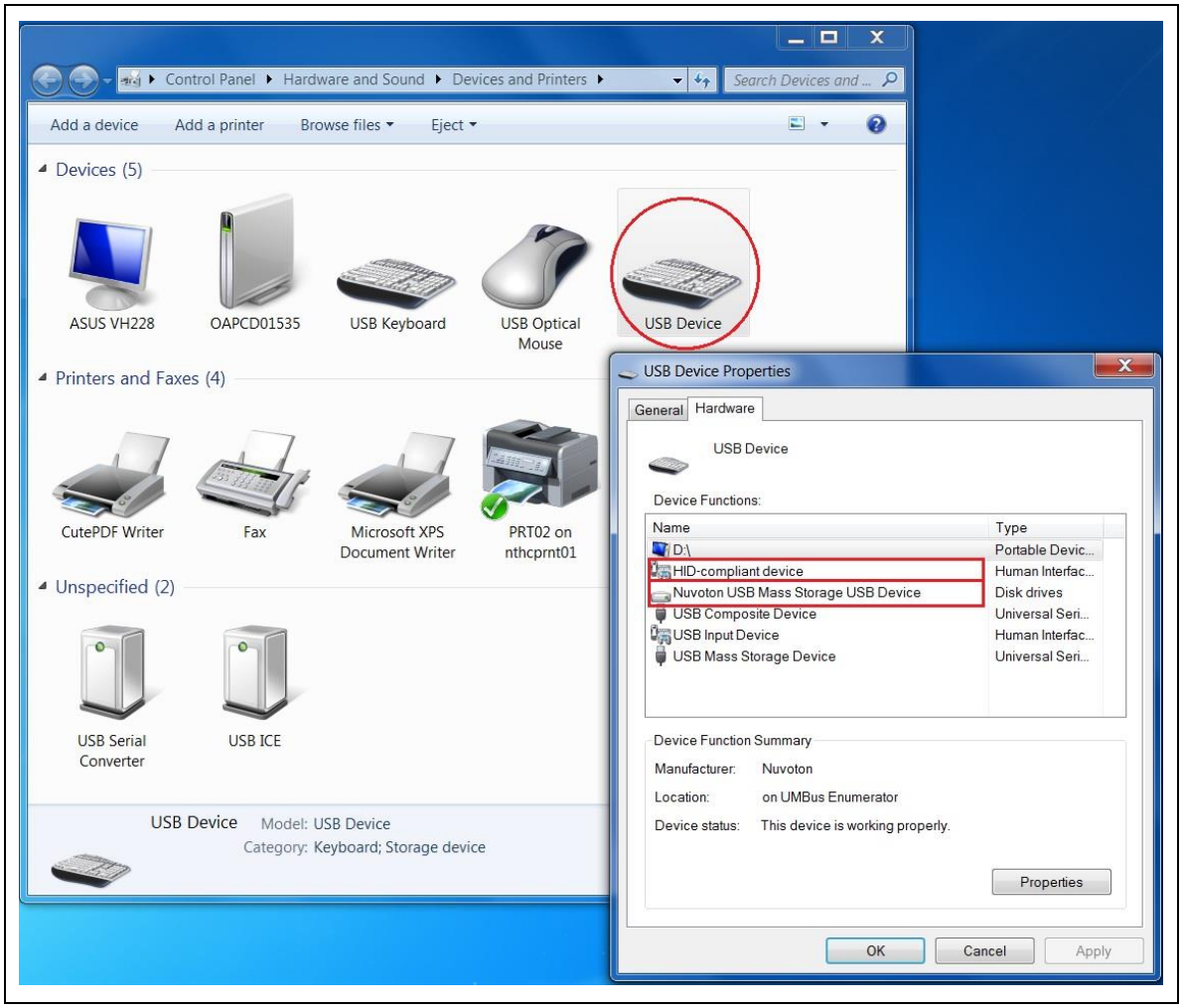


圖 8 HID Transfer+MSC的複合裝置

3.3.2 HID Transfer + HID Keyboard

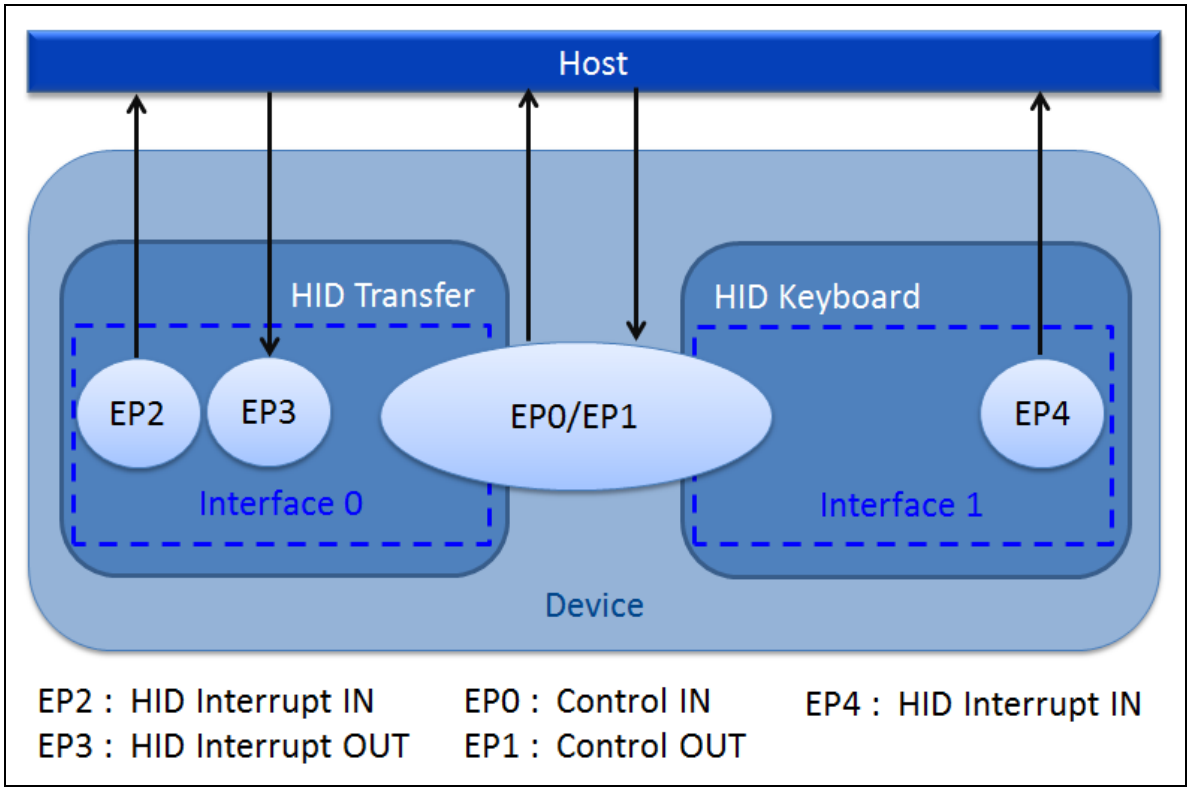


圖 9 HID Transfer + HID Keyboard複合裝置的架構

這個複合裝置範例是結合HID Transfer和HID Keyboard，圖 9說明HID Transfer和HID Keyboard複合裝置範例的架構。首先可以先從BSP\SampleCode\StdDriver分別取得HID Transfer和HID Keyboard這兩個獨立功能的範例程式，使用者可以藉由使用HID Transfer的範例程式並結合HID Keyboard的範例程式來實作出新的HID Transfer和HID Keyboard的複合裝置範例。開發步驟如下：

步驟1：根據裝置的類別來配置端點和其對應的IN/OUT Buffer，此範例配置五個端點EP0~EP4。EP0_BUF_BASE~EP4_BUF_BASE分別為各個端點可以使用的Buffer起始位址。EP0被配置為Control IN，EP1被配置為Control OUT，EP0和EP1的最大封包長度皆為8Bytes。EP2被配置為Interrupt IN，EP3被配置為Interrupt OUT，最大封包長度皆為64Bytes，EP4被配置為Interrupt IN，最大封包長度皆為8Bytes。下面的程式碼顯示Endpoint的設定。

```
/* 端點最大封包長度 */
#define EP0_MAX_PKT_SIZE    8
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    64
#define EP3_MAX_PKT_SIZE    64
#define EP4_MAX_PKT_SIZE    8
```

```
/* 各個端點在USB Buffer的偏移值 */
#define SETUP_BUF_BASE  0
#define SETUP_BUF_LEN   8
#define EP0_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP0_BUF_LEN      EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP1_BUF_LEN      EP1_MAX_PKT_SIZE
#define EP2_BUF_BASE    (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN      EP2_MAX_PKT_SIZE
#define EP3_BUF_BASE    (EP2_BUF_BASE + EP2_BUF_LEN)
#define EP3_BUF_LEN      EP3_MAX_PKT_SIZE
#define EP4_BUF_BASE    (EP3_BUF_BASE + EP3_BUF_LEN)
#define EP4_BUF_LEN      EP4_MAX_PKT_SIZE

void HID_Init(void)
{
    /* Init setup packet buffer */
    /* Buffer range for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;

    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

    /*****************************************************/
    /* EP2 ==> Interrupt IN endpoint, address 1 */
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
    /* Buffer range for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

    /* EP3 ==> Interrupt OUT endpoint, address 2 */
```

```
    USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_OUT | INT_OUT_EP_NUM);
    /* Buffer range for EP3 */
    USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);
    /* trigger to receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);

    /*****************************************************/
    /* EP4 ==> Interrupt IN endpoint, address 3 */
    USBD_CONFIG_EP(EP4, USBD_CFG_EPMODE_IN | HID_KB_EP_NUM);
    /* Buffer range for EP4 */
    USBD_SET_EP_BUF_ADDR(EP4, EP4_BUF_BASE);
}
```

步驟2：設備、配置、介面、端點和字串描述元都包含在"descriptor.c"中，使用者在此檔案中加入HID Transfer和HID Keyboard這兩個類別的各種描述元。

```
/* USB設備描述元， VID和PID決定USB主機要安裝哪個驅動程式 */
uint8_t gu8DeviceDescriptor[] = {
    LEN_DEVICE,      /* bLength */
    DESC_DEVICE,     /* bDescriptorType */
    0x10, 0x01,      /* bcdUSB */
    0x00,            /* bDeviceClass */
    0x00,            /* bDeviceSubClass */
    0x00,            /* bDeviceProtocol */
    EP0_MAX_PKT_SIZE,   /* bMaxPacketSize0 */
    /* idVendor */
    USBD_VID & 0x00FF,
    (USBD_VID & 0xFF00) >> 8,
    /* idProduct */
    USBD_PID & 0x00FF,
    (USBD_PID & 0xFF00) >> 8,
    0x00, 0x00,      /* bcdDevice */
    0x01,            /* iManufacture */
    0x02,            /* iProduct */
    0x02,            /* iSerialNumber */
    0x01             /* bNumConfigurations */
};
/* USB 組態描述元，它包含介面、HID及端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
/* 組態描述元 */
```

```
    LEN_CONFIG,     /* bLength */
    DESC_CONFIG,    /* bDescriptorType */
    // wTotalLength
    LEN_CONFIG_AND_SUBORDINATE & 0x00FF,
    (LEN_CONFIG_AND_SUBORDINATE & 0xFF00) >> 8,
    0x02,           /* bNumInterfaces */
    0x01,           /* bConfigurationValue */
    0x00,           /* iConfiguration */
    0x80 | (USBD_SELF_POWERED << 6) | (USBD_REMOTE_WAKEUP << 5),/*
bmAttributes */
    USBD_MAX_POWER,         /* MaxPower */

/* 介面0 : HID Transfer介面描述元 */
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
    0x00,           /* bInterfaceNumber */
    0x00,           /* bAlternateSetting */
    0x02,           /* bNumEndpoints */
    0x03,           /* bInterfaceClass */
    0x00,           /* bInterfaceSubClass */
    0x00,           /* bInterfaceProtocol */
    0x00,           /* iInterface */

/* HID描述元 */
    LEN_HID,        /* Size of this descriptor in UINT8s. */
    DESC_HID,       /* HID descriptor type. */
    0x10, 0x01,     /* HID Class Spec. release number. */
    0x00,           /* H/W target country. */
    0x01,           /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,   /* Descriptor type. */
    /* Total length of report descriptor. */
    sizeof(HID_DeviceReportDescriptor) & 0x00FF,
    (sizeof(HID_DeviceReportDescriptor) & 0xFF00) >> 8,

/* 端點2描述元: 中斷IN，地址為INT_IN_EP_NUM */
    LEN_ENDPOINT,                               /* bLength */
    DESC_ENDPOINT,                              /* bDescriptorType */
    (INT_IN_EP_NUM | EP_INPUT),                 /* bEndpointAddress */
    EP_INT,                                     /* bmAttributes */
```

```
    /* wMaxPacketSize */
    EP2_MAX_PKT_SIZE & 0x00FF,
    (EP2_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,                /* bInterval */

/* 端點3描述元: 中斷OUT，地址為INT_OUT_EP_NUM */
    LEN_ENDPOINT,                               /* bLength */
    DESC_ENDPOINT,                              /* bDescriptorType */
    (INT_OUT_EP_NUM | EP_OUTPUT),               /* bEndpointAddress */
    EP_INT,                                     /* bmAttributes */
    /* wMaxPacketSize */
    EP3_MAX_PKT_SIZE & 0x00FF,
    (EP3_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,                /* bInterval */

/* 介面1 : HID Keyboard介面描述元 */
    LEN_INTERFACE,                      /* bLength */
    DESC_INTERFACE,                     /* bDescriptorType */
    0x01,                               /* bInterfaceNumber */
    0x00,                               /* bAlternateSetting */
    0x01,                               /* bNumEndpoints */
    0x03,                               /* bInterfaceClass */
    0x00,                               /* bInterfaceSubClass */
    HID_KEYBOARD,                       /* bInterfaceProtocol */
    0x00,                               /* iInterface */

/* HID描述元 */
    LEN_HID,        /* Size of this descriptor in UINT8s. */
    DESC_HID,       /* HID descriptor type. */
    0x10, 0x01,     /* HID Class Spec. release number. */
    0x00,           /* H/W target country. */
    0x01,           /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,   /* Descriptor type. */
    /* Total length of report descriptor. */
    sizeof(HID_KeyboardReportDescriptor) & 0x00FF,
    (sizeof(HID_KeyboardReportDescriptor) & 0xFF00) >> 8,

/* 端點4描述元: 中斷IN，地址為HID_KB_EP_NUM */
    LEN_ENDPOINT,                       /* bLength */
```

```
    DESC_ENDPOINT,                      /* bDescriptorType */
    (HID_KB_EP_NUM | EP_INPUT),         /* bEndpointAddress */
    EP_INT,                             /* bmAttributes */
    /* wMaxPacketSize */
    EP4_MAX_PKT_SIZE & 0x00FF,
    (EP4_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL         /* bInterval */
};

/* HID Transfer Report描述元 */
uint8_t HID_DeviceReportDescriptor[] =
{
    0x06, 0x00, 0xFF,   // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,         // Usage (Vendor Usage 1)
    0xA1, 0x01,         // Collection (Application)
    0x19, 0x01,         // Usage Minimum
    0x29, 0x40,         // Usage Maximum
    0x15, 0x00,         // Logical Minimum
    0x26, 0xFF, 0x00,   // Logical Maximum
    0x75, 0x08,         // Report Size: 8-bit field size
    0x95, 0x40,         // Report Count
    // an "Input", "Output", or "Feature" item)
    0x81, 0x00,         // Input (Data, Array, Abs)
   // above report size, count, logical min/max, and usage.
    0x19, 0x01,         // Usage Minimum
    0x29, 0x40,         // Usage Maximum
    0x91, 0x00,         // Output (Data, Array, Abs)
    0xC0                // End Collection
};
/* HID Keyboard報告描述元 */
uint8_t HID_KeyboardReportDescriptor[] =
{
    0x05, 0x01,         /* Usage Page(Generic Desktop Controls) */
    0x09, 0x06,         /* Usage(Keyboard) */
    0xA1, 0x01,         /* Collection(Application) */
    0x05, 0x07,         /* Usage Page(Keyboard/Keypad) */
    0x19, 0xE0,         /* Usage Minimum(0xE0) */
    0x29, 0xE7,         /* Usage Maximum(0xE7) */
    0x15, 0x00,         /* Logical Minimum(0x0) */
```

```
    0x25, 0x01,         /* Logical Maximum(0x1) */
    0x75, 0x01,         /* Report Size(0x1) */
    0x95, 0x08,         /* Report Count(0x8) */
    0x81, 0x02,         /* Input (Data) => Modifier byte */
    0x95, 0x01,         /* Report Count(0x1) */
    0x75, 0x08,         /* Report Size(0x8) */
    0x81, 0x01,         /* Input (Constant) => Reserved byte */
    0x95, 0x05,         /* Report Count(0x5) */
    0x75, 0x01,         /* Report Size(0x1) */
    0x05, 0x08,         /* Usage Page(LEDs) */
    0x19, 0x01,         /* Usage Minimum(0x1) */
    0x29, 0x05,         /* Usage Maximum(0x5) */
    0x91, 0x02,         /* Output (Data) => LED report */
    0x95, 0x01,         /* Report Count(0x1) */
    0x75, 0x03,         /* Report Size(0x3) */
    0x91, 0x01,         /* Output (Constant) => LED report padding */
    0x95, 0x06,         /* Report Count(0x6) */
    0x75, 0x08,         /* Report Size(0x8) */
    0x15, 0x00,         /* Logical Minimum(0x0) */
    0x25, 0x65,         /* Logical Maximum(0x65) */
    0x05, 0x07,         /* Usage Page(Keyboard/Keypad) */
    0x19, 0x00,         /* Usage Minimum(0x0) */
    0x29, 0x65,         /* Usage Maximum(0x65) */
    0x81, 0x00,         /* Input (Data) */
    0xC0,               /* End Collection */
};

/* 字串描述元 */
uint8_t gu8StringLang[4] = {
    4,                                          /* bLength */
    DESC_STRING,                                /* bDescriptorType */
    0x09, 0x04
};

/* 字串描述元 */
uint8_t gu8VendorStringDesc[] = {
    16,
    DESC_STRING,
    'N', 0, 'u', 0, 'v', 0, 'o', 0, 't', 0, 'o', 0, 'n', 0
```

```
};

/* 字串描述元 */
uint8_t gu8ProductStringDesc[] = {
    22,                                         /* bLength          */
    DESC_STRING,                                /* bDescriptorType  */
    'U', 0, 'S', 0, 'B', 0, ' ', 0, 'D', 0, 'e', 0, 'v', 0, 'i', 0, 'c', 0,
'e', 0
};

/* 因為此範例的HID為介面0和介面1，因此HID相關結構的Index分別為0和1 */
uint8_t *gu8UsbHidReport[3] = {
    HID_DeviceReportDescriptor,             /* HID transfer報告描述元 */
    HID_KeyboardReportDescriptor,           /* HID keyboard報告描述元 */
    NULL,
};

uint32_t gu32UsbHidReportLen[3] = {
    sizeof(HID_DeviceReportDescriptor),     /* HID transfer報告描述元的長度 */
    sizeof(HID_KeyboardReportDescriptor),   /* HID keyboard報告描述元的長度 */
    0,
};

uint32_t gu32ConfigHidDescIdx[3] = {
/* HID transfer描述元在組態描述元內的位置 */
    (LEN_CONFIG+LEN_INTERFACE),
/* HID keyboard描述元在組態描述元內的位置 */
    (sizeof(gu8ConfigDescriptor) - LEN_HID - LEN_ENDPOINT),
    0,
};
```

步驟3：hid_transfer_And_Keyboard.c執行HID類別請求和端點資料處理，USB在HID設備中定義了六個類別請求，但是並不是所有的命令都必須回應。hid_transfer_And_Keyboard.c執行HID類別請求及端點EP2~EP4的Callback如下所示：

```
void HID_ClassRequest(void)
{
    uint8_t buf[8];
    USBD_GetSetupPacket(buf);
```

```
if (buf[0] & 0x80) {              /* 該命令數據傳輸方向是Device - > Host */
    switch (buf[1]) {
          switch (buf[1]) {
    case GET_REPORT:                                          /* 不處理 */
    case GET_IDLE:                                            /* 不處理 */
    case GET_PROTOCOL:                                        /* 不處理 */
    default: {
        /* Setup error, stall the device */
        USBD_SetStall(0);
        break;
    }
    }
} else {
    /* Host -> Device */
    switch (buf[1]) {
    case SET_REPORT: {                                  /* HID的類別請求 */
       if (buf[3] == 3) {
            /* Request Type = Feature */
            USBD_SET_DATA1(EP1);
            USBD_SET_PAYLOAD_LEN(EP1, 0);
        }
        else if(buf[3] == 2)
        {
            /* Request Type = Output */
            USBD_SET_DATA1(EP1);
            USBD_SET_PAYLOAD_LEN(EP1, buf[6]);

            /* Trigger for HID Int in */
            USBD_SET_PAYLOAD_LEN(EP4, 0);

            /* Status stage */
            USBD_PrepareCtrlIn(0, 0);
        }
        break;
    }
    case SET_IDLE: {                                    /* HID的類別請求 */
        /* Status stage */
        USBD_SET_DATA1(EP0);
```

```
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;
        }
        case SET_PROTOCOL:                                          /* 不處理 */
        default: {
            // Stall
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
        }
    }
}

void EP2_Handler(void)  /* Interrupt IN handler */
{
    HID_SetInReport();
}

void EP3_Handler(void)  /* Interrupt OUT handler */
{
    uint8_t *ptr;
    /* Interrupt OUT */
    ptr = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP3));
    HID_GetOutReport(ptr, USBD_GET_PAYLOAD_LEN(EP3));
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
}

void EP4_Handler(void)  /* Interrupt IN handler */
{
    g_u8EP4Ready = 1;
}
```

執行此範例，當裝置列舉成功時，可在電腦的裝置管理員中看到HID Transfer+HID Keyboard的複合裝置，不需另外安裝驅動程式，如圖 10所示。HID Transfer可透過PC端的Windows Tool來與HID裝置互相通訊；另一個HID Keyboard則模擬在鍵盤上輸入按鍵'A'。

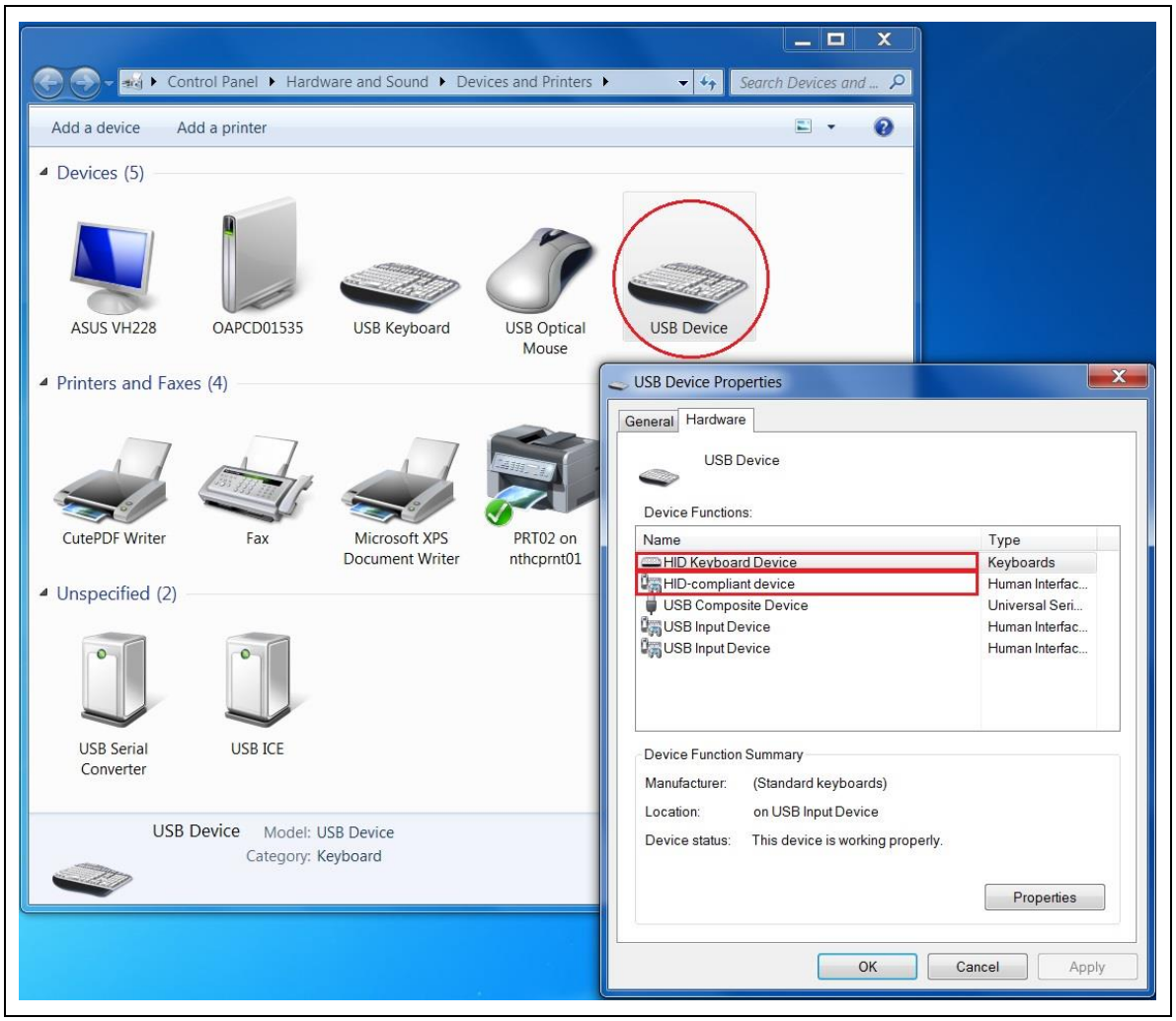


圖 10 HID Transfer + HID Keyboard 複合裝置

3.3.3 HID Mouse + HID Keyboard

這個複合裝置範例是結合HID Mouse和HID Keyboard，圖 11說明HID Mouse和HID Keyboard複合裝置範例的架構。首先可以先從BSP\SampleCode\StdDriver分別取得HID Mouse和HID Keyboard這兩個獨立功能的範例程式，使用者可以藉由使用HID Mouse的範例程式並結合HID Keyboard的範例程式來實作出新的HID Mouse和HID Keyboard的複合裝置範例。

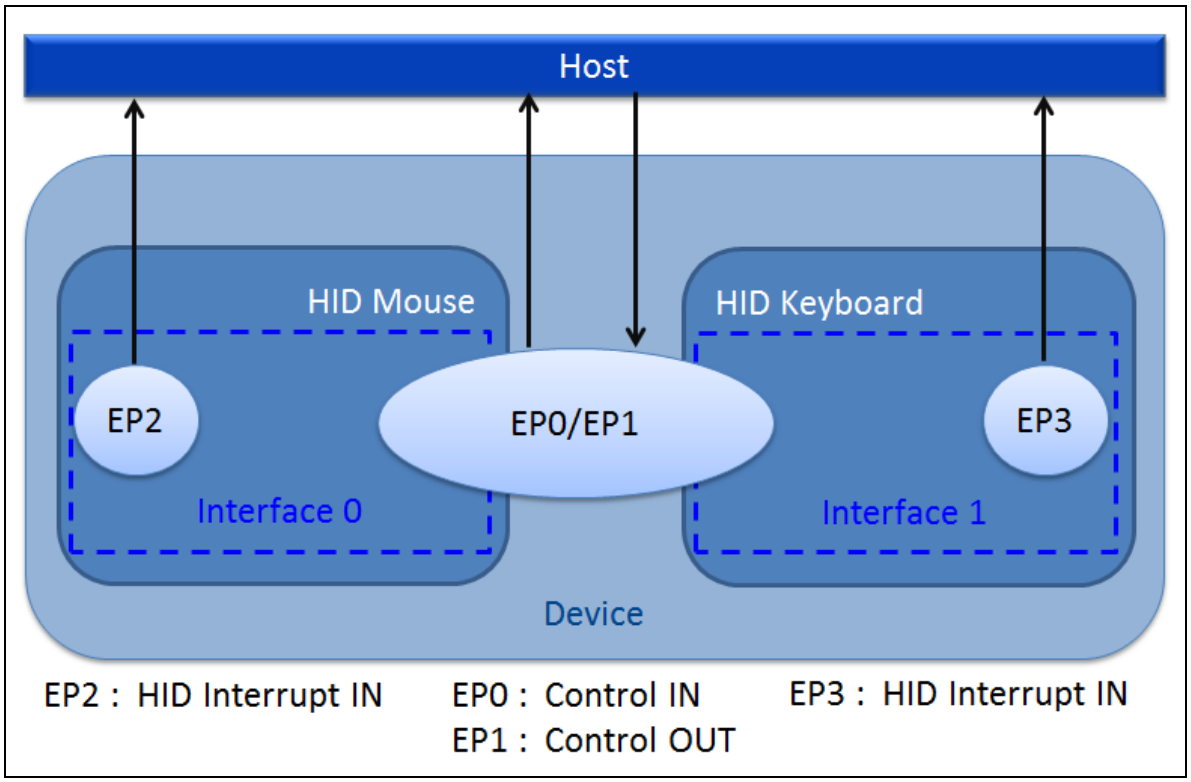


圖 11 HID Mouse + HID Keyboard複合裝置的架構

開發步驟如下：

步驟1：根據裝置的類別來配置端點和其對應的IN/OUT Buffer，此範例配置四個端點EP0~EP3。EP0_BUF_BASE~EP3_BUF_BASE分別為各個端點可以使用的Buffer起始位址。EP0被配置為Control IN，EP1被配置為Control OUT，EP2被配置為Interrupt IN，EP3被配置為Interrupt IN，最大封包長度皆為8Bytes。下面的程式碼顯示Endpoint的設定。

```
/* 端點最大封包長度 */
#define EP0_MAX_PKT_SIZE    8
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    8
#define EP3_MAX_PKT_SIZE    8

/* 各個端點在USB Buffer的偏移值 */
#define SETUP_BUF_BASE  0
#define SETUP_BUF_LEN   8
#define EP0_BUF_BASE     (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP0_BUF_LEN      EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
```

```
#define EP1_BUF_LEN       EP1_MAX_PKT_SIZE
#define EP2_BUF_BASE    (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN       EP2_MAX_PKT_SIZE
#define EP3_BUF_BASE    (EP2_BUF_BASE + EP2_BUF_LEN)
#define EP3_BUF_LEN       EP3_MAX_PKT_SIZE

void HID_Init(void)
{
    /* Init setup packet buffer */
    /* Buffer range for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;

    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

    /*****************************************************/
    /* EP2 ==> Interrupt IN endpoint, address 1 */
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | HID_MOUSE_EP_NUM);
    /* Buffer range for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

    /*****************************************************/
    /* EP3 ==> Interrupt IN endpoint, address 2 */
    USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_IN | HID_KB_EP_NUM);
    /* Buffer range for EP3 */
    USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);
}
```

步驟2：設備、配置、介面、端點和字串描述元都包含在"descriptor.c"中，使用者在此檔案中加入HID Mouse和HID Keyboard這兩個類別的各種描述元。

```
/* USB設備描述元， VID和PID決定USB主機要安裝哪個驅動程式 */
uint8_t gu8DeviceDescriptor[] = {
    LEN_DEVICE,     /* bLength */
    DESC_DEVICE,    /* bDescriptorType */
    0x10, 0x01,     /* bcdUSB */
    0x00,           /* bDeviceClass */
    0x00,           /* bDeviceSubClass */
    0x00,           /* bDeviceProtocol */
    EP0_MAX_PKT_SIZE,   /* bMaxPacketSize0 */
    /* idVendor */
    USBD_VID & 0x00FF,
    (USBD_VID & 0xFF00) >> 8,
    /* idProduct */
    USBD_PID & 0x00FF,
    (USBD_PID & 0xFF00) >> 8,
    0x00, 0x00,     /* bcdDevice */
    0x01,           /* iManufacture */
    0x02,           /* iProduct */
    0x02,           /* iSerialNumber */
    0x01            /* bNumConfigurations */
};
/* USB 組態描述元，它包含介面、HID及端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
/* 組態描述元 */
    LEN_CONFIG,                                     /* bLength */
    DESC_CONFIG,                                    /* bDescriptorType */
    // wTotalLength
    LEN_CONFIG_AND_SUBORDINATE & 0x00FF,
    (LEN_CONFIG_AND_SUBORDINATE & 0xFF00) >> 8,
    0x02,                                           /* bNumInterfaces */
    0x01,                                           /* bConfigurationValue */
    0x00,                                           /* iConfiguration */
    /* bmAttributes */
    0x80 | (USBD_SELF_POWERED << 6) | (USBD_REMOTE_WAKEUP << 5),
    USBD_MAX_POWER,                                 /* MaxPower */

/* 介面0 : HID Mouse介面描述元 */
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
```

```
    0x00,           /* bInterfaceNumber */
    0x00,           /* bAlternateSetting */
    0x01,           /* bNumEndpoints */
    0x03,           /* bInterfaceClass */
    0x00,           /* bInterfaceSubClass */
    0x00,           /* bInterfaceProtocol */
    0x00,           /* iInterface */

/* HID描述元 */
    LEN_HID,        /* Size of this descriptor in UINT8s. */
    DESC_HID,       /* HID descriptor type. */
    0x10, 0x01,     /* HID Class Spec. release number. */
    0x00,           /* H/W target country. */
    0x01,           /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,   /* Descriptor type. */
    /* Total length of report descriptor. */
    sizeof(HID_MouseReportDescriptor) & 0x00FF,
    (sizeof(HID_MouseReportDescriptor) & 0xFF00) >> 8,

/* 端點2描述元: 中斷IN，地址為HID_MOUSE_EP_NUM */
    LEN_ENDPOINT,                       /* bLength */
    DESC_ENDPOINT,                      /* bDescriptorType */
    (HID_MOUSE_EP_NUM | EP_INPUT),      /* bEndpointAddress */
    EP_INT,                             /* bmAttributes */
    /* wMaxPacketSize */
    EP2_MAX_PKT_SIZE & 0x00FF,
    (EP2_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,        /* bInterval */

/* 介面1 : HID Keyboard介面描述元*/
    LEN_INTERFACE,                      /* bLength */
    DESC_INTERFACE,                     /* bDescriptorType */
    0x01,                               /* bInterfaceNumber */
    0x00,                               /* bAlternateSetting */
    0x01,                               /* bNumEndpoints */
    0x03,                               /* bInterfaceClass */
    0x00,                               /* bInterfaceSubClass */
    HID_KEYBOARD,                       /* bInterfaceProtocol */
    0x00,                               /* iInterface */
```

```
/* HID描述元 */
    LEN_HID,        /* Size of this descriptor in UINT8s. */
    DESC_HID,       /* HID descriptor type. */
    0x10, 0x01,     /* HID Class Spec. release number. */
    0x00,           /* H/W target country. */
    0x01,           /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,   /* Descriptor type. */
    /* Total length of report descriptor. */
    sizeof(HID_KeyboardReportDescriptor) & 0x00FF,
    (sizeof(HID_KeyboardReportDescriptor) & 0xFF00) >> 8,

/* 端點3描述元: 中斷IN，地址為HID_KB_EP_NUM */
    LEN_ENDPOINT,                /* bLength */
    DESC_ENDPOINT,               /* bDescriptorType */
    (HID_KB_EP_NUM | EP_INPUT),  /* bEndpointAddress */
    EP_INT,                      /* bmAttributes */
    /* wMaxPacketSize */
    EP3_MAX_PKT_SIZE & 0x00FF,
    (EP3_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL       /* bInterval */
};

/* HID Mouse報告描述元 */
uint8_t HID_MouseReportDescriptor[] = {
    0x05, 0x01,             /* Usage Page(Generic Desktop Controls) */
    0x09, 0x02,             /* Usage(Mouse) */
    0xA1, 0x01,             /* Collection(Application) */
    0x09, 0x01,             /* Usage(Pointer) */
    0xA1, 0x00,               /* Collection(Physical) */
    0x05, 0x09,             /* Usage Page(Button) */
    0x19, 0x01,             /* Usage Minimum(0x1) */
    0x29, 0x03,             /* Usage Maximum(0x3) */
    0x15, 0x00,             /* Logical Minimum(0x0) */
    0x25, 0x01,             /* Logical Maximum(0x1) */
    0x75, 0x01,             /* Report Size(0x1) */
    0x95, 0x03,             /* Report Count(0x3) */
    0x81, 0x02,             /* Input(3 button bit) */
    0x75, 0x05,             /* Report Size(0x5) */
```

```
    0x95, 0x01,             /* Report Count(0x1) */
    0x81, 0x01,             /* Input(5 bit padding) */
    0x05, 0x01,             /* Usage Page(Generic Desktop Controls) */
    0x09, 0x30,             /* Usage(X) */
    0x09, 0x31,             /* Usage(Y) */
    0x09, 0x38,             /* Usage(Wheel) */
    0x15, 0x81,             /* Logical Minimum(0x81)(-127) */
    0x25, 0x7F,             /* Logical Maximum(0x7F)(127) */
    0x75, 0x08,             /* Report Size(0x8) */
    0x95, 0x03,             /* Report Count(0x3) */
    0x81, 0x06,             /* Input(1 byte wheel) */
    0xC0,                   /* End Collection */
    0xC0,                   /* End Collection */
};

/* HID Keyboard報告描述元 */
uint8_t HID_KeyboardReportDescriptor[] =
{
    0x05, 0x01,         /* Usage Page(Generic Desktop Controls) */
    0x09, 0x06,         /* Usage(Keyboard) */
    0xA1, 0x01,         /* Collection(Application) */
    0x05, 0x07,         /* Usage Page(Keyboard/Keypad) */
    0x19, 0xE0,         /* Usage Minimum(0xE0) */
    0x29, 0xE7,         /* Usage Maximum(0xE7) */
    0x15, 0x00,         /* Logical Minimum(0x0) */
    0x25, 0x01,         /* Logical Maximum(0x1) */
    0x75, 0x01,         /* Report Size(0x1) */
    0x95, 0x08,         /* Report Count(0x8) */
    0x81, 0x02,         /* Input (Data) => Modifier byte */
    0x95, 0x01,         /* Report Count(0x1) */
    0x75, 0x08,         /* Report Size(0x8) */
    0x81, 0x01,         /* Input (Constant) => Reserved byte */
    0x95, 0x05,         /* Report Count(0x5) */
    0x75, 0x01,         /* Report Size(0x1) */
    0x05, 0x08,         /* Usage Page(LEDs) */
    0x19, 0x01,         /* Usage Minimum(0x1) */
    0x29, 0x05,         /* Usage Maximum(0x5) */
    0x91, 0x02,         /* Output (Data) => LED report */
    0x95, 0x01,         /* Report Count(0x1) */
```

```
    0x75, 0x03,         /* Report Size(0x3) */
    0x91, 0x01,         /* Output (Constant) => LED report padding */
    0x95, 0x06,         /* Report Count(0x6) */
    0x75, 0x08,         /* Report Size(0x8) */
    0x15, 0x00,         /* Logical Minimum(0x0) */
    0x25, 0x65,         /* Logical Maximum(0x65) */
    0x05, 0x07,         /* Usage Page(Keyboard/Keypad) */
    0x19, 0x00,         /* Usage Minimum(0x0) */
    0x29, 0x65,         /* Usage Maximum(0x65) */
    0x81, 0x00,         /* Input (Data) */
    0xC0                /* End Collection */
};

/* 字串描述元 */
uint8_t gu8StringLang[4] = {
    4,                                          /* bLength */
    DESC_STRING,                                /* bDescriptorType */
    0x09, 0x04
};

/* 字串描述元 */
uint8_t gu8VendorStringDesc[] = {
    16,
    DESC_STRING,
    'N', 0, 'u', 0, 'v', 0, 'o', 0, 't', 0, 'o', 0, 'n', 0
};

/* 字串描述元 */
uint8_t gu8ProductStringDesc[] = {
    24,
    DESC_STRING,
    'N', 0, 'u', 0, 'M', 0, 'i', 0, 'c', 0, 'r', 0, 'o', 0, ' ', 0, 'H', 0,
'I', 0, 'D', 0
};

/* 因為此範例的HID為介面0和介面1，因此HID相關結構的Index分別為0和1 */
uint8_t *gu8UsbHidReport[3] = {
    HID_MouseReportDescriptor,              /* HID Mouse報告描述元 */
    HID_KeyboardReportDescriptor,           /* HID keyboard報告描述元 */
```

```
    NULL,
};

uint32_t gu32UsbHidReportLen[3] = {
    sizeof(HID_MouseReportDescriptor),     /* HID Mouse報告描述元的長度 */
    sizeof(HID_KeyboardReportDescriptor),  /* HID keyboard報告描述元的長度 */
    0,
};

uint32_t gu32ConfigHidDescIdx[3] = {
/* HID Mouse描述元在組態描述元內的位置 */
    (LEN_CONFIG+LEN_INTERFACE),
/* HID keyboard描述元在組態描述元內的位置 */
    (sizeof(gu8ConfigDescriptor) - LEN_HID - LEN_ENDPOINT),
    0,
};
```

步驟3：hid_mousekeyboard.c執行HID類別請求和端點資料處理，USB在HID設備中定義了六個類別請求，但是並不是所有的命令都必須回應。hid_mousekeyboard.c執行HID類別請求及端點EP2和EP3的Callback如下所示：

```
void HID_ClassRequest(void)
{
    uint8_t buf[8];
    USBD_GetSetupPacket(buf);

    if (buf[0] & 0x80) {              /* 該命令數據傳輸方向是Device -> Host */
        switch (buf[1]) {
        case GET_REPORT:                                          /* 不處理 */
        case GET_IDLE:                                            /* 不處理 */
        case GET_PROTOCOL:                                        /* 不處理 */
        default: {
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
        }
    } else {
        /* Host -> Device */
        switch (buf[1]) {
```

```
        case SET_REPORT: {                                  /* HID的類別請求 */
            if (buf[3] == 3)
            {
                /* Request Type = Feature */
                USBD_SET_DATA1(EP1);
                USBD_SET_PAYLOAD_LEN(EP1, 0);
            }
            else if(buf[3] == 2)
            {
                /* Request Type = Output */
                USBD_SET_DATA1(EP1);
                USBD_SET_PAYLOAD_LEN(EP1, buf[6]);

                /* Status stage */
                USBD_PrepareCtrlIn(0, 0);
            }
            break;
        }
        case SET_IDLE: {                                    /* HID的類別請求 */
            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;
        }
        case SET_PROTOCOL:                                          /* 不處理 */
        default: {
            // Stall
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
        }
    }
}

void EP2_Handler(void)  /* Interrupt IN handler */
{
    g_u8EP2Ready = 1;
}
```

```
void EP3_Handler(void)  /* Interrupt IN handler */
{
     g_u8EP3Ready = 1;
}
```

執行此範例當裝置列舉成功時，可在電腦的裝置管理員中看到HID Mouse + HID Keyboard的複合裝置，不需另外安裝驅動程式，如圖 12所示。HID Mouse模擬滑鼠在電腦螢幕上畫圓移動顯示；另一個HID Keyboard則模擬在鍵盤上輸入按鍵'A'。

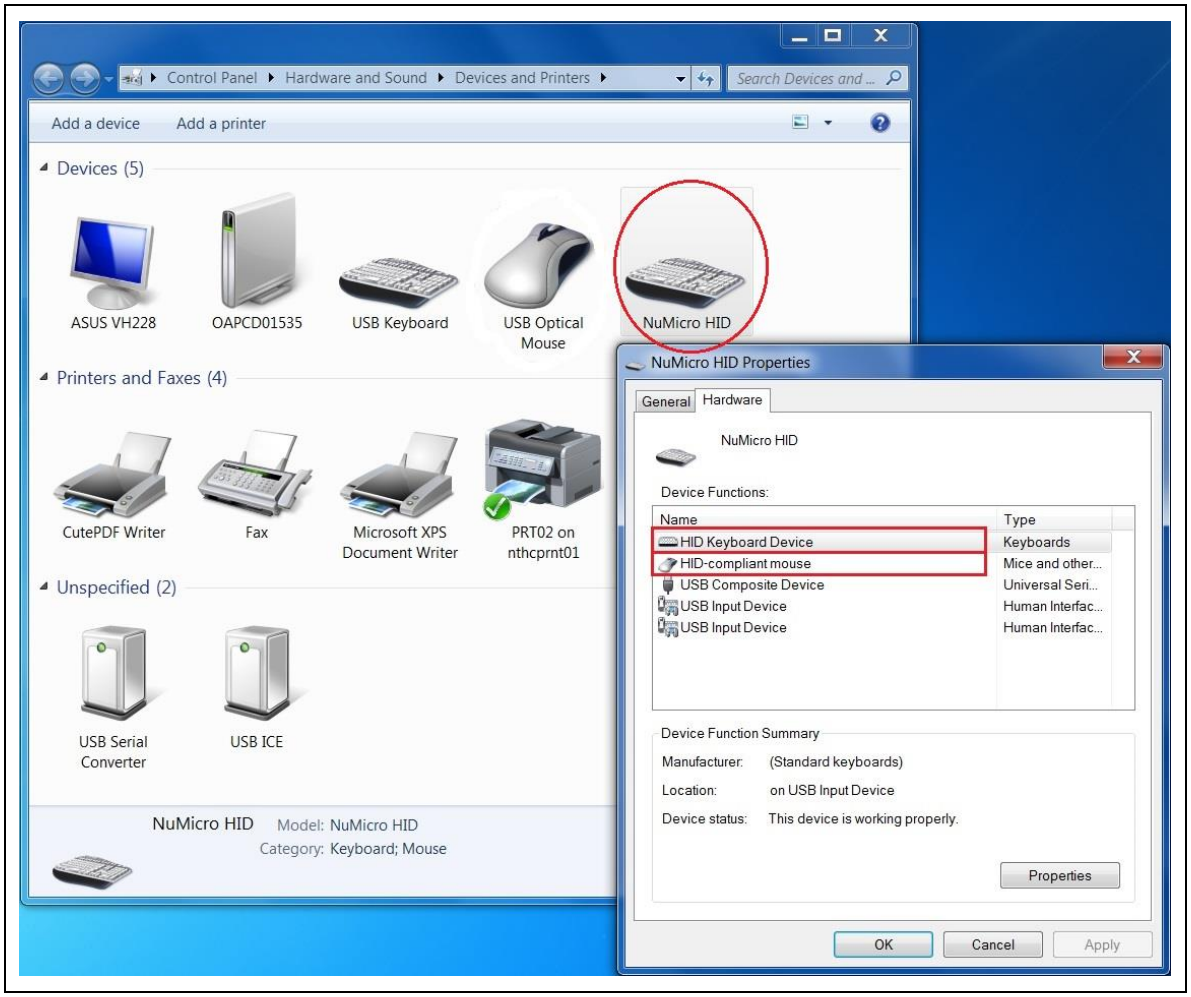


圖 12 HID Mouse + HID Keyboard 複合裝置

3.3.4 VCOM + HID Transfer

VCOM和HID Transfer組成。圖 13說明VCOM + HID Transfer複合裝置範例的架構。首先可以先從BSP\SampleCode\StdDriver分別取得VCOM和HID Transfer這兩個獨立功能的範例程式，使用者可以藉由使用VCOM的範例程式並結合HID

Transfer的範例程式來實作出新的VCOM和HID Transfer的複合裝置範例。

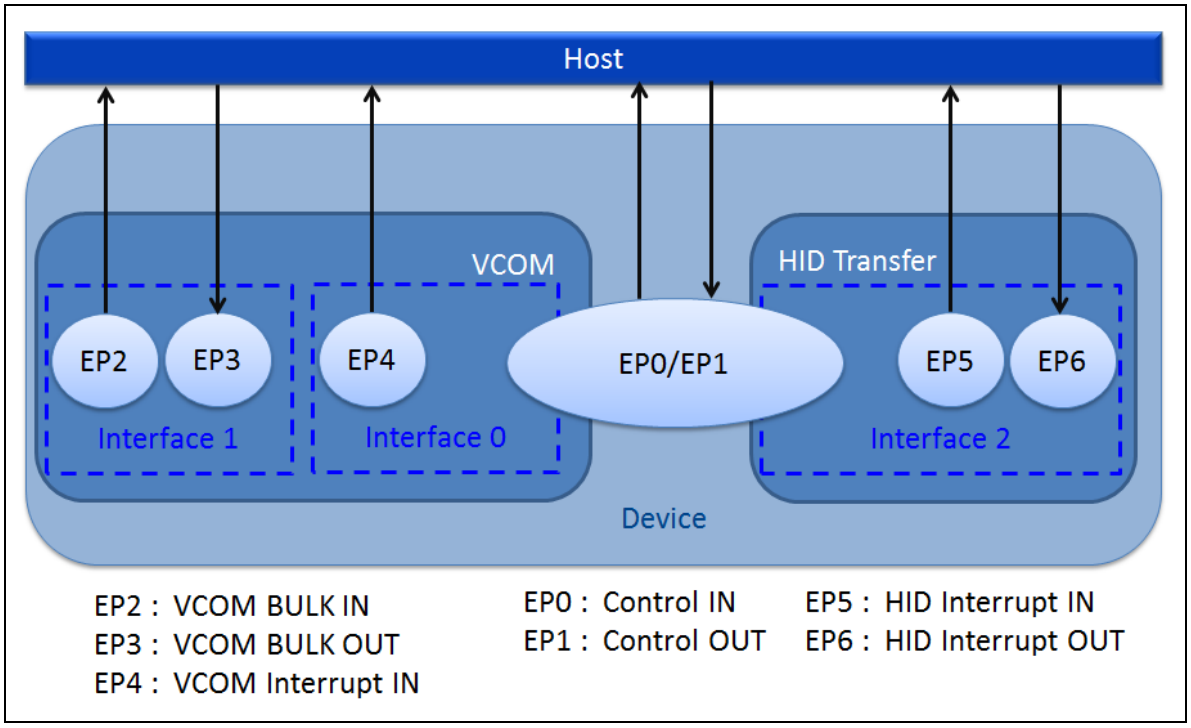


圖 13 VCOM + HID Transfer複合裝置的架構

開發步驟如下：

步驟1：根據裝置的類別來配置端點和其對應的IN/OUT Buffer，此範例配置七個端點EP0~EP6。EP0_BUF_BASE~EP6_BUF_BASE分別為各個端點可以使用的Buffer起始位址。EP0被配置為Control IN，EP1被配置為Control OUT，EP2被配置為BULK IN，EP3被配置為BULK OUT，EP4被配置為Interrupt IN，EP5被配置為Interrupt IN，EP6被配置為Interrupt OUT，除了EP0、EP1及EP4的最大封包長度為8Bytes，其他端點的最大封包長度皆為64Bytes。下面的程式碼顯示Endpoint的設定。

```
/* 端點最大封包長度 */
#define EP0_MAX_PKT_SIZE    8
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    64
#define EP3_MAX_PKT_SIZE    64
#define EP4_MAX_PKT_SIZE    8
#define EP5_MAX_PKT_SIZE    64
#define EP6_MAX_PKT_SIZE    64

/* 各個端點在USB Buffer的偏移值 */
```

```
#define SETUP_BUF_BASE  0
#define SETUP_BUF_LEN   8
#define EP0_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP0_BUF_LEN      EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP1_BUF_LEN      EP1_MAX_PKT_SIZE
#define EP2_BUF_BASE    (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN      EP2_MAX_PKT_SIZE
#define EP3_BUF_BASE    (EP2_BUF_BASE + EP2_BUF_LEN)
#define EP3_BUF_LEN      EP3_MAX_PKT_SIZE
#define EP4_BUF_BASE    (EP3_BUF_BASE + EP3_BUF_LEN)
#define EP4_BUF_LEN      EP4_MAX_PKT_SIZE
#define EP5_BUF_BASE    (EP4_BUF_BASE + EP4_BUF_LEN)
#define EP5_BUF_LEN      EP5_MAX_PKT_SIZE
#define EP6_BUF_BASE    (EP5_BUF_BASE + EP5_BUF_LEN)
#define EP6_BUF_LEN      EP6_MAX_PKT_SIZE

void HID_Init(void)
{
    /* Init setup packet buffer */
    /* Buffer range for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;

    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

    /*****************************************************/
    /* EP2 ==> Bulk IN endpoint, address 1 */
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | BULK_IN_EP_NUM);
    /* Buffer offset for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);
```

```
    /* EP3 ==> Bulk Out endpoint, address 2 */
    USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_OUT | BULK_OUT_EP_NUM);
    /* Buffer offset for EP3 */
    USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);
    /* trigger receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);

    /* EP4 ==> Interrupt IN endpoint, address 3 */
    USBD_CONFIG_EP(EP4, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
    /* Buffer offset for EP4 */
    USBD_SET_EP_BUF_ADDR(EP4, EP4_BUF_BASE);

    /*******************************************************/
    /* EP5 ==> Interrupt IN endpoint, address 4 */
    USBD_CONFIG_EP(EP5, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM_1);
    /* Buffer range for EP5 */
    USBD_SET_EP_BUF_ADDR(EP5, EP5_BUF_BASE);

    /* EP6 ==> Interrupt OUT endpoint, address 5 */
    USBD_CONFIG_EP(EP6, USBD_CFG_EPMODE_OUT | INT_OUT_EP_NUM_1);
    /* Buffer range for EP6 */
    USBD_SET_EP_BUF_ADDR(EP6, EP6_BUF_BASE);
    /* trigger to receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP6, EP6_MAX_PKT_SIZE);
}
```

步驟2：設備、配置、介面、端點和字串描述元都包含在"descriptor.c"中，使用者在此檔案中加入VCOM和HID Transfer這兩個類別的各種描述元。因為此範例屬於多重介面的複合裝置，需要額外使用介面關聯描述元(IAD)來定義多重介面功能，因此設備描述元中的三個欄位(bDeviceClass、bDeviceSubClass及bDeviceProtocol)的值分別為：0xEF, 0x02, 0x01。

```
/* USB設備描述元， VID和PID決定USB主機要安裝哪個驅動程式 */
uint8_t gu8DeviceDescriptor[] = {
    LEN_DEVICE,         /* bLength */
    DESC_DEVICE,        /* bDescriptorType */
    0x00, 0x02,         /* bcdUSB */
    0xEF,               /* bDeviceClass: IAD*/
    0x02,               /* bDeviceSubClass */
```

```
    0x01,               /* bDeviceProtocol */
    EP0_MAX_PKT_SIZE,   /* bMaxPacketSize0 */
    /* idVendor */
    USBD_VID & 0x00FF,
    (USBD_VID & 0xFF00) >> 8,
    /* idProduct */
    USBD_PID & 0x00FF,
    (USBD_PID & 0xFF00) >> 8,
    0x00, 0x03,     /* bcdDevice */
    0x01,           /* iManufacture */
    0x02,           /* iProduct */
    0x00,           /* iSerialNumber - no serial */
    0x01            /* bNumConfigurations */
}

/* USB 組態描述元，它包含介面、HID及端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
/* 組態描述元 */
    LEN_CONFIG,         /* bLength              */
    DESC_CONFIG,        /* bDescriptorType      */
    0x6B, 0x00,         /* wTotalLength         */
    0x03,               /* bNumInterfaces       */
    0x01,               /* bConfigurationValue  */
    0x00,               /* iConfiguration       */
    0xC0,               /* bmAttributes         */
    0x32,               /* MaxPower             */

/* 介面關聯描述元(IAD) */
    0x08,               // bLength: Interface Descriptor size
    0x0B,               // bDescriptorType: IAD
    0x00,               // bFirstInterface
    0x02,               // bInterfaceCount
    0x02,               // bFunctionClass: CDC
    0x02,               // bFunctionSubClass
    0x01,               // bFunctionProtocol
    0x02,               // iFunction

/* 介面0 : VCOM Control介面描述元 */
    LEN_INTERFACE,  /* bLength              */
```

```
    DESC_INTERFACE, /* bDescriptorType      */
    0x00,           /* bInterfaceNumber     */
    0x00,           /* bAlternateSetting    */
    0x01,           /* bNumEndpoints        */
    0x02,           /* bInterfaceClass      */
    0x02,           /* bInterfaceSubClass   */
    0x01,           /* bInterfaceProtocol   */
    0x00,           /* iInterface           */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x00,           /* Header functional descriptor subtype */
    0x10, 0x01,     /* Communication device compliant to the communication
spec. ver. 1.10 */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x01,           /* Call management functional descriptor */
    0x00,           /* BIT0: Whether device handle call management itself. */
                    /* BIT1: Whether device can send/receive call management
information over a Data Class Interface 0 */
    0x01,           /* Interface number of data class interface optionally
used for call management */

    /* Communication Class Specified INTERFACE descriptor */
    0x04,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x02,           /* Abstract control management functional descriptor
subtype */
    0x00,           /* bmCapabilities       */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* bLength              */
    0x24,           /* bDescriptorType: CS_INTERFACE descriptor type */
    0x06,           /* bDescriptorSubType   */
    0x00,           /* bMasterInterface     */
    0x01,           /* bSlaveInterface0     */
```

```
/* 端點4描述元: 中斷IN，地址為INT_IN_EP_NUM */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_INPUT | INT_IN_EP_NUM),     /* bEndpointAddress */
    EP_INT,                         /* bmAttributes     */
    EP4_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x01,                           /* bInterval        */

/* 介面1 : VCOM Data介面描述元 */
    LEN_INTERFACE,                  /* bLength              */
    DESC_INTERFACE,                 /* bDescriptorType      */
    0x01,                           /* bInterfaceNumber     */
    0x00,                           /* bAlternateSetting    */
    0x02,                           /* bNumEndpoints        */
    0x0A,                           /* bInterfaceClass      */
    0x00,                           /* bInterfaceSubClass   */
    0x00,                           /* bInterfaceProtocol   */
    0x00,                           /* iInterface           */

/* 端點2描述元: BULK IN，地址為BULK_IN_EP_NUM */
    /* ENDPOINT descriptor */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_INPUT | BULK_IN_EP_NUM),    /* bEndpointAddress */
    EP_BULK,                        /* bmAttributes     */
    EP2_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x00,                           /* bInterval        */

/* 端點3描述元: BULK OUT，地址為BULK_OUT_EP_NUM */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_OUTPUT | BULK_OUT_EP_NUM),  /* bEndpointAddress */
    EP_BULK,                        /* bmAttributes     */
    EP3_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x00,                           /* bInterval        */

/* 介面2 : HID Transfer介面描述元 */
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
```

```
    0x02,           /* bInterfaceNumber */
    0x00,           /* bAlternateSetting */
    0x02,           /* bNumEndpoints */
    0x03,           /* bInterfaceClass */
    0x00,           /* bInterfaceSubClass */
    0x00,           /* bInterfaceProtocol */
    0x00,           /* iInterface */

/* HID描述元 */
    LEN_HID,        /* Size of this descriptor in UINT8s. */
    DESC_HID,       /* HID descriptor type. */
    0x10, 0x01,     /* HID Class Spec. release number. */
    0x00,           /* H/W target country. */
    0x01,           /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,   /* Descriptor type. */
    /* Total length of report descriptor. */
    sizeof(HID_DeviceReportDescriptor) & 0x00FF,
    (sizeof(HID_DeviceReportDescriptor) & 0xFF00) >> 8,

/* 端點5描述元: 中斷IN，地址為INT_IN_EP_NUM_1 */
    LEN_ENDPOINT,                       /* bLength */
    DESC_ENDPOINT,                      /* bDescriptorType */
    (INT_IN_EP_NUM_1 | EP_INPUT),       /* bEndpointAddress */
    EP_INT,                             /* bmAttributes */
    /* wMaxPacketSize */
    EP5_MAX_PKT_SIZE & 0x00FF,
    (EP5_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,        /* bInterval */

/* 端點6描述元: 中斷OUT，地址為INT_OUT_EP_NUM_1 */
    LEN_ENDPOINT,                       /* bLength */
    DESC_ENDPOINT,                      /* bDescriptorType */
    (INT_OUT_EP_NUM_1 | EP_OUTPUT),     /* bEndpointAddress */
    EP_INT,                             /* bmAttributes */
    /* wMaxPacketSize */
    EP6_MAX_PKT_SIZE & 0x00FF,
    (EP6_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL,        /* bInterval */
};
```

```
/* HID Transfer Report描述元 */
uint8_t HID_DeviceReportDescriptor[] =
{
    0x06, 0x00, 0xFF,   // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,         // Usage (Vendor Usage 1)
    0xA1, 0x01,         // Collection (Application)
    0x19, 0x01,         // Usage Minimum
    0x29, 0x40,         // Usage Maximum
    0x15, 0x00,         // Logical Minimum
    0x26, 0xFF, 0x00,   // Logical Maximum
    0x75, 0x08,         // Report Size: 8-bit field size
    0x95, 0x40,         // Report Count
    // an "Input", "Output", or "Feature" item)
    0x81, 0x00,         // Input (Data, Array, Abs)
   // above report size, count, logical min/max, and usage.
    0x19, 0x01,         // Usage Minimum
    0x29, 0x40,         // Usage Maximum
    0x91, 0x00,         // Output (Data, Array, Abs)
    0xC0                // End Collection
};

/* 字串描述元 */
uint8_t gu8StringLang[4] = {
    4,                                          /* bLength */
    DESC_STRING,                                /* bDescriptorType */
    0x09, 0x04
};

/* 字串描述元 */
uint8_t gu8VendorStringDesc[] = {
    16,
    DESC_STRING,
    'N', 0, 'u', 0, 'v', 0, 'o', 0, 't', 0, 'o', 0, 'n', 0
};

/* 字串描述元 */
uint8_t gu8ProductStringDesc[] = {
    22,             /* bLength          */
```

```
    DESC_STRING,    /* bDescriptorType  */
    'U', 0, 'S', 0, 'B', 0, ' ', 0, 'D', 0, 'e', 0, 'v', 0, 'i', 0, 'c', 0,
'e', 0
};

/* 因為此範例的HID為介面2，因此HID相關結構的Index皆為2 */
uint8_t *gu8UsbHidReport[3] = {
    NULL,
    NULL,
    HID_DeviceReportDescriptor,            /* HID transfer報告描述元 */
};

uint32_t gu32UsbHidReportLen[3] = {
    0,
    0,
    sizeof(HID_DeviceReportDescriptor),    /* HID transfer報告描述元的長度 */
};

uint32_t gu32ConfigHidDescIdx[3] = {
    0,
    0,
/* HID transfer描述元在組態描述元內的位置 */
    (sizeof(gu8ConfigDescriptor) - LEN_HID - (2*LEN_ENDPOINT)),
};
```

步驟3：VCOM_And_hid_transfer.c執行VCOM/HID類別請求和端點資料處理，在USB通信設備類別（CDC）的抽象控制模式中定義虛擬COM埠設備必要的特殊類別請求有三個，HID設備定義六個類別請求，但是並不是所有的命令都必須回應。VCOM_And_hid_transfer.c執行VCOM/HID類別請求及端點EP2~EP6的Callback如下所示：

```
void HID_ClassRequest(void)
{
    uint8_t buf[8];
    USBD_GetSetupPacket(buf);

    if (buf[0] & 0x80) {                /* 該命令數據傳輸方向是Device -> Host */
        switch (buf[1]) {
         case GET_LINE_CODE: {  /* 介面0: VCOM的類別請求 */
                if (buf[4] == 0) { /* VCOM-1 */
```

```
                    USBD_MemCopy((uint8_t *)(USBD_BUF_BASE +
USBD_GET_EP_BUF_ADDR(EP0)), (uint8_t *)&gLineCoding, 7);
                }
                /* Data stage */
                USBD_SET_DATA1(EP0);
                USBD_SET_PAYLOAD_LEN(EP0, 7);
                /* Status stage */
                USBD_PrepareCtrlOut(0,0);
                break;
            }
        /* 介面0: HID的類別請求 */
        case GET_REPORT:                                        /* 不處理 */
        case GET_IDLE:                                          /* 不處理 */
        case GET_PROTOCOL:                                      /* 不處理 */
        default: {
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
        }
    }
    else {
    /* Host -> Device */
    switch (buf[1]) {
       case SET_CONTROL_LINE_STATE: {           /* 介面0: VCOM的類別請求 */
            if (buf[4] == 0) { /* VCOM-1 */
                gCtrlSignal = buf[3];
                gCtrlSignal = (gCtrlSignal << 8) | buf[2];
            }

            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;
        }
        case SET_LINE_CODE: {                   /* 介面0: VCOM的類別請求 */
            if (buf[4] == 0) /* VCOM-1 */
                USBD_PrepareCtrlOut((uint8_t *)&gLineCoding, 7);
```

```
            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);

            break;
        }
        /* 介面2: HID的類別請求 */
        case SET_REPORT: {
            if (buf[3] == 3) {
                /* Request Type = Feature */
                USBD_SET_DATA1(EP1);
                USBD_SET_PAYLOAD_LEN(EP1, 0);
            }
            break;
        }
        /* 介面2: HID的類別請求 */
        case SET_IDLE: {
            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;
        }
        /* 介面2: HID的類別請求 */
        case SET_PROTOCOL:                                          /* 不處理 */
        default: {
            // Stall
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
        }
    }
}

void EP2_Handler(void)  /* BULK IN handler */
{
    gu32TxSize = 0;
}
```

```
void EP3_Handler(void)  /* BULK OUT handler */
{
    /* Bulk OUT */
    gu32RxSize = USBD_GET_PAYLOAD_LEN(EP3);
    gpu8RxBuf = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP3));

    /* Set a flag to indicate bulk out ready */
    gi8BulkOutReady = 1;
}

void EP5_Handler(void)  /* Interrupt IN handler */
{
    HID_SetInReport();
}

void EP6_Handler(void)  /* Interrupt OUT handler */
{
    uint8_t *ptr;
    /* Interrupt OUT */
    ptr = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP6));
    HID_GetOutReport(ptr, USBD_GET_PAYLOAD_LEN(EP6));
    USBD_SET_PAYLOAD_LEN(EP6, EP6_MAX_PKT_SIZE);
}
```

第一次執行此範例時，必須先安裝驅動程式，否則電腦會無法辨識此設備，如圖 14所示。透過此範例搭配的INF檔案來安裝CDC驅動程式(圖 15)，成功安裝驅動程式後，電腦的裝置管理員就會出現一個新的COM Port代表安裝完成，如圖 16所示。系統會自動為它指定一個可用的埠號，本範例埠號被指定為COM 51。使用者可以利用網路上免費的終端機連線工具來與COM Port進行資料傳輸。圖 17為執行VCOM範例的執行結果，使用時先設定終端機連線工具及選擇所對應的COM Port，使用者可以在終端機連線工具上看到自己從鍵盤輸入的特定字元。

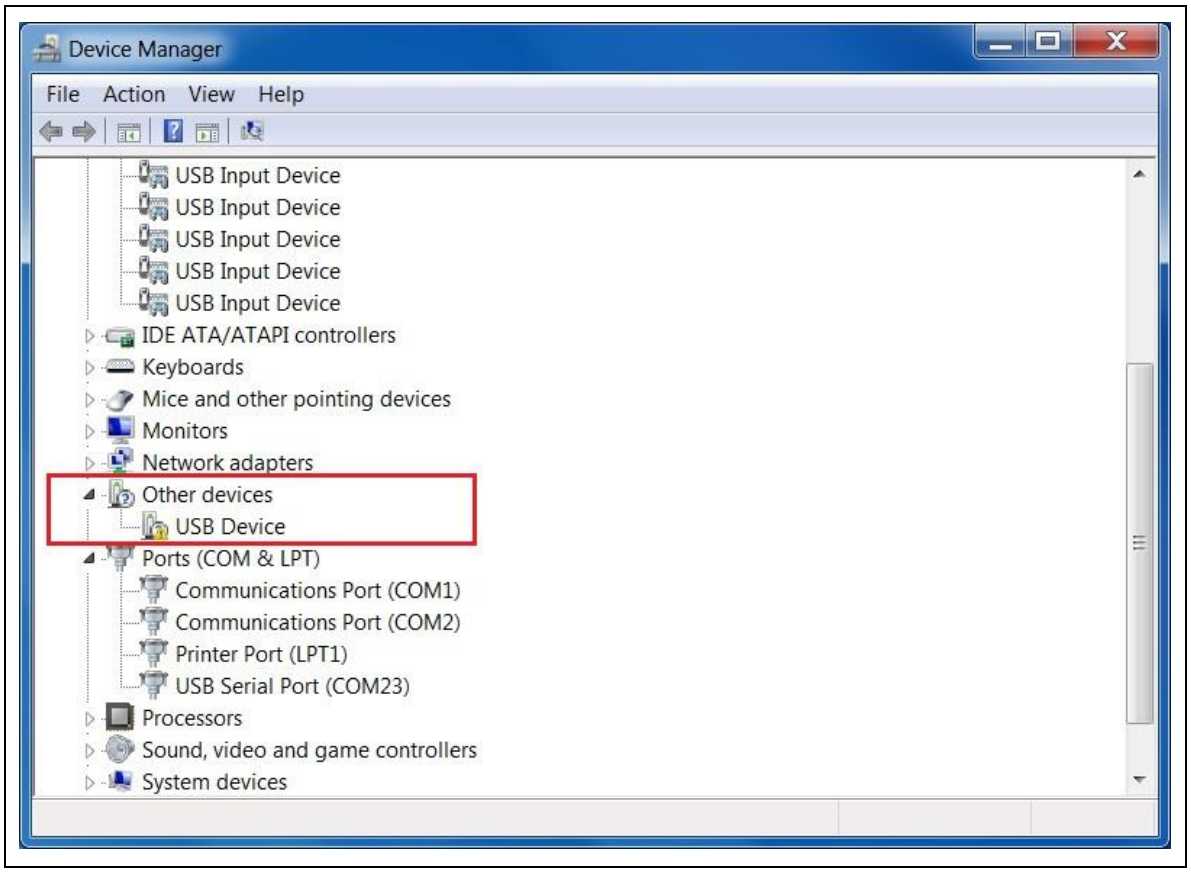
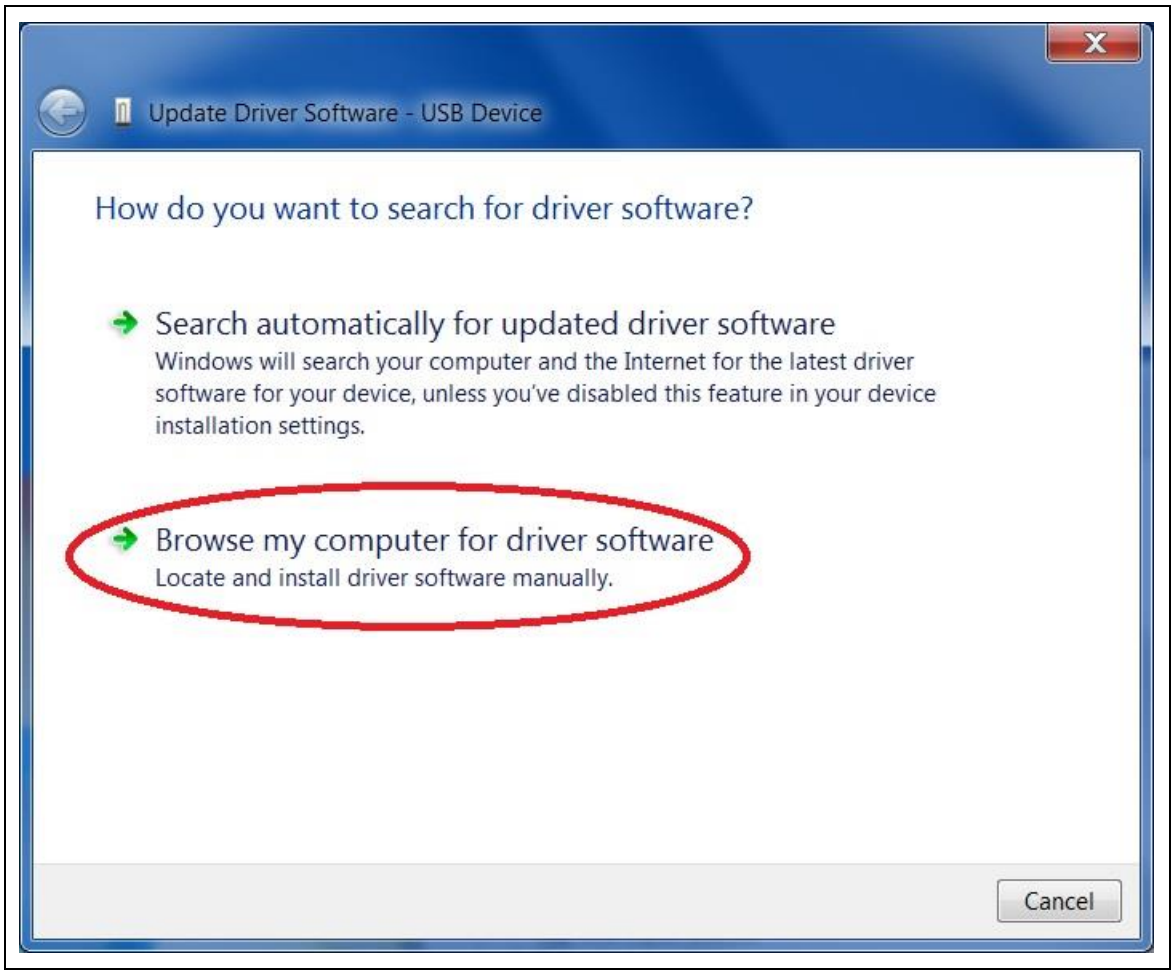


圖 14 未安裝驅動程式(VCOM + HID Transfer複合裝置)


Update Driver Software - USB Device
How do you want to search for driver software?
Search automatically for updated driver software
Windows will search your computer and the Internet for the latest driver software for your device, unless you've disabled this feature in your device installation settings.
Browse my computer for driver software
Locate and install driver software manually.
Cancel

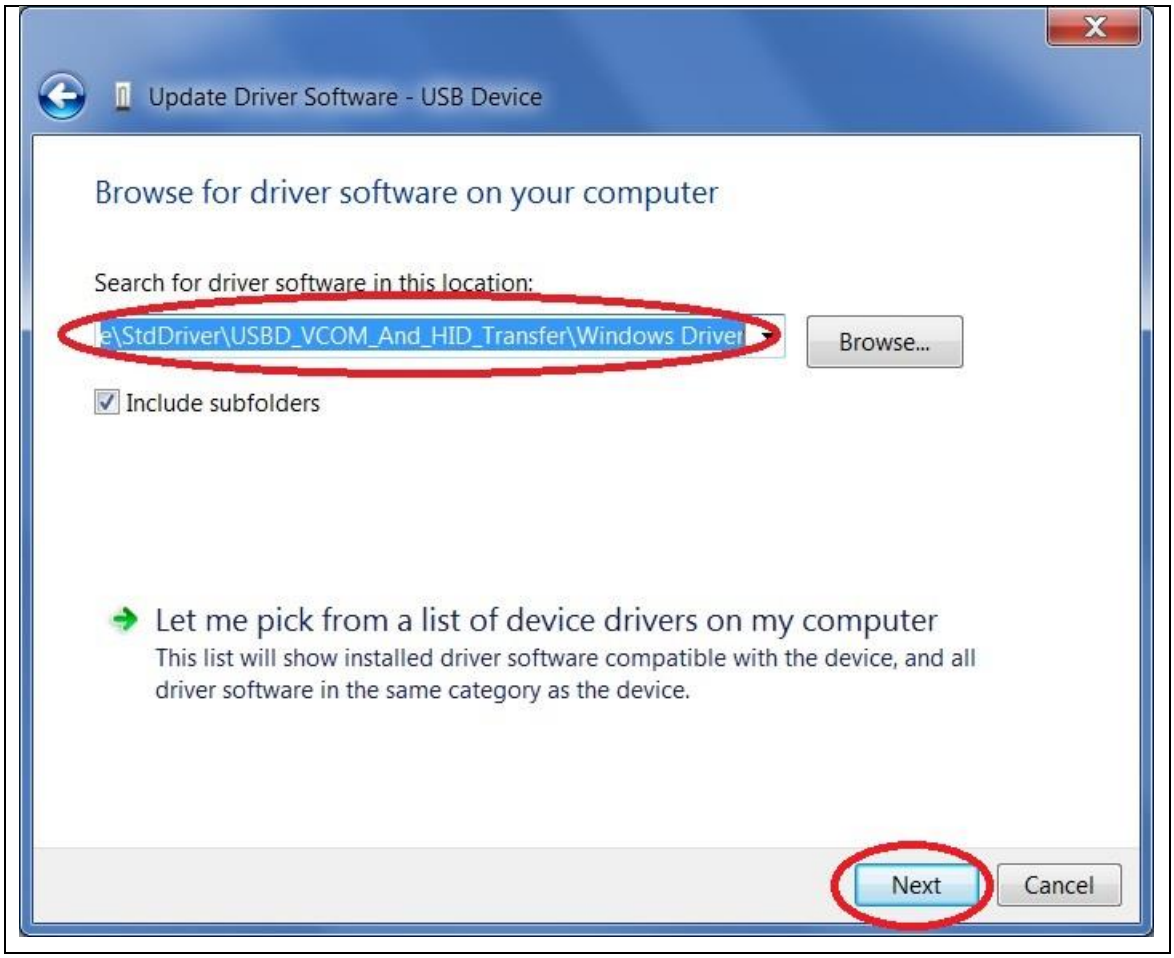

Update Driver Software - USB Device
Browse for driver software on your computer
Search for driver software in this location:
e\StdDriver\USBD_VCOM_And_HID_Transfer\Windows Driver
Browse...
Include subfolders
Let me pick from a list of device drivers on my computer
This list will show installed driver software compatible with the device, and all driver software in the same category as the device.
Next
Cancel

圖 15 安裝INF

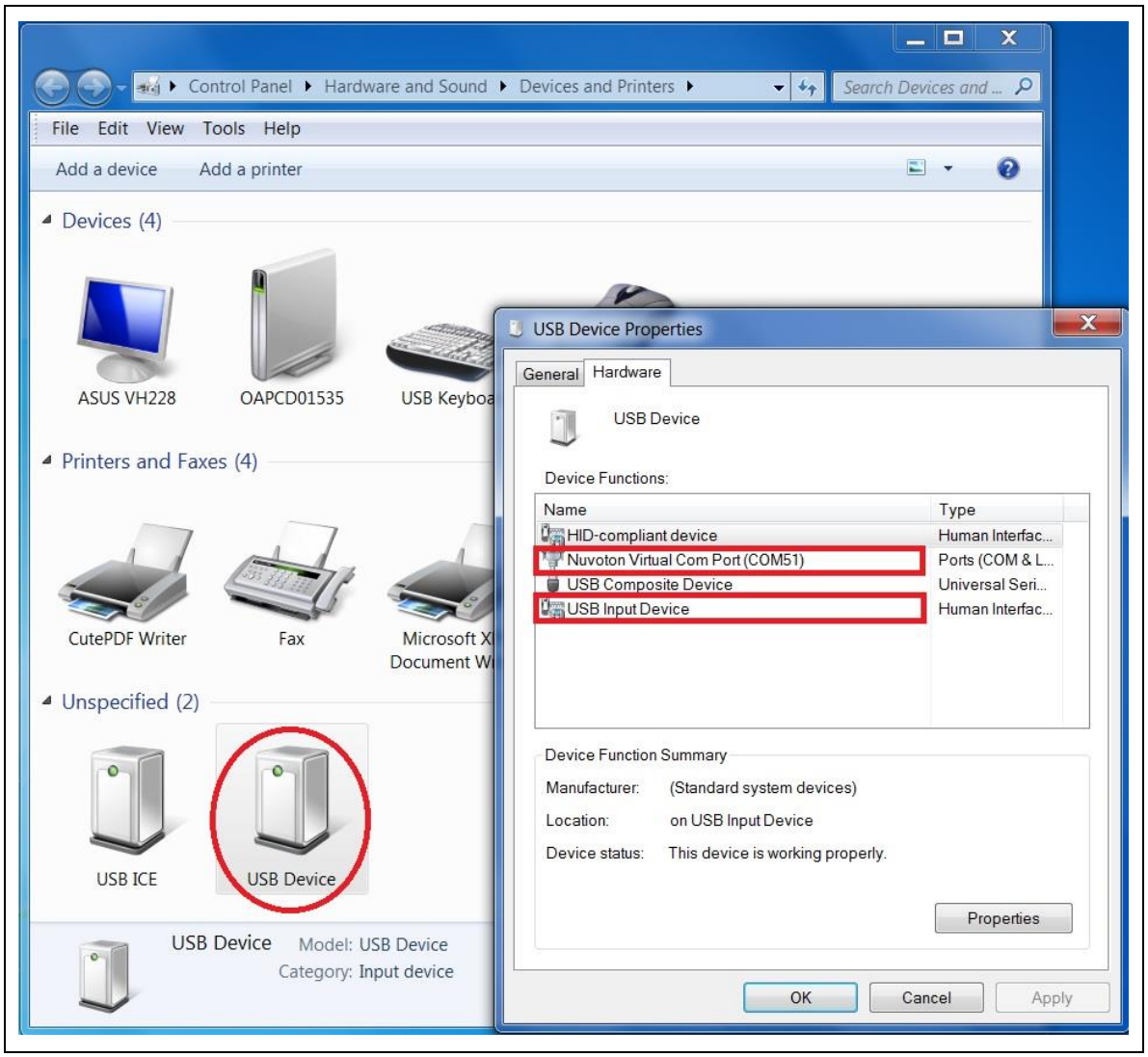


圖 16 VCOM + HID Transfer複合裝置

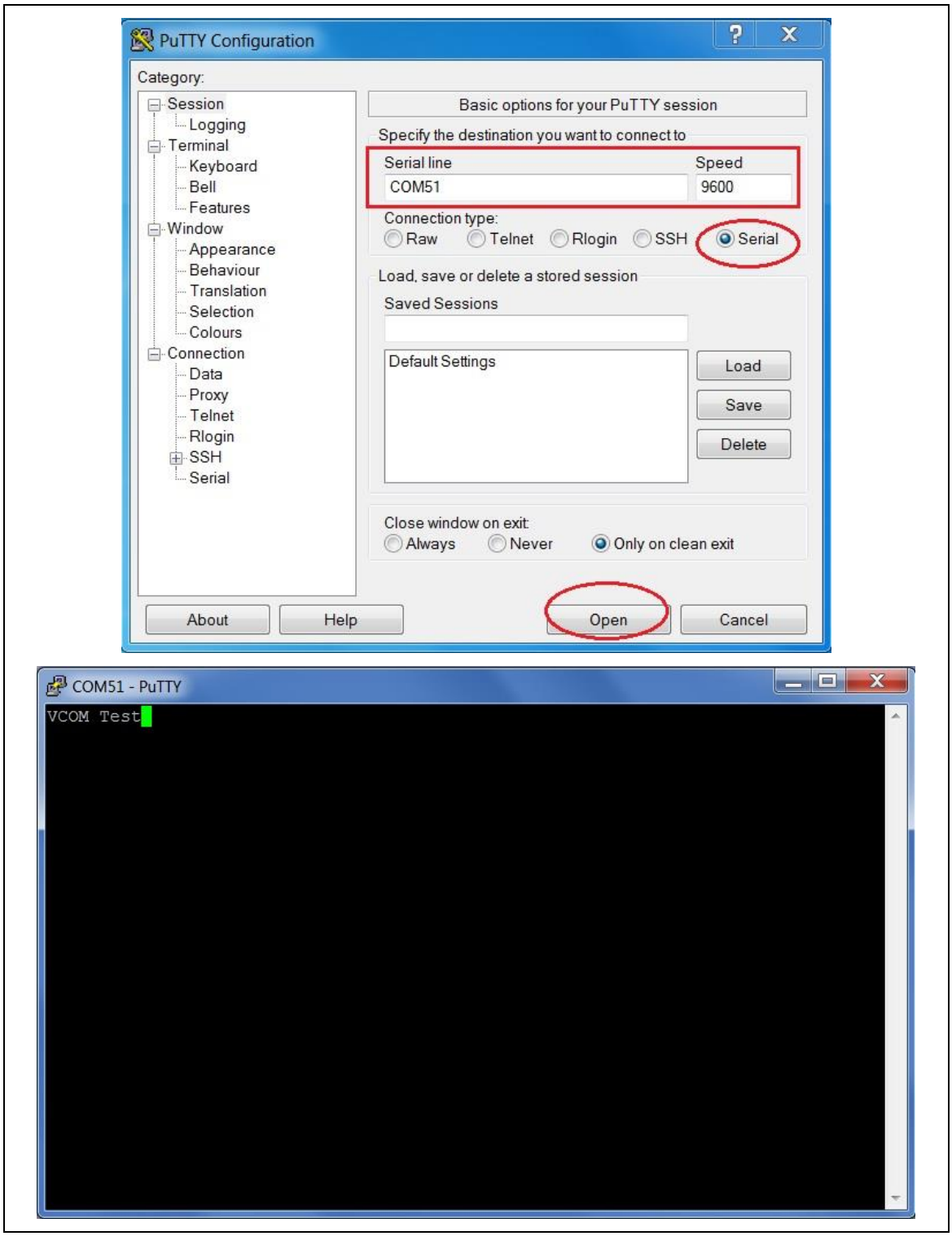


圖 17 終端機設定及VCOM 範例

3.3.5 VCOM + MSC

VCOM和MSC組成。圖 18說明VCOM + MSC複合裝置範例的架構。首先可以先從BSP\SampleCode\StdDriver分別取得VCOM和MSC這兩個獨立功能的範例程式，使用者可以藉由使用VCOM的範例程式並結合MSC的範例程式來實作出新的VCOM和MSC的複合裝置範例。

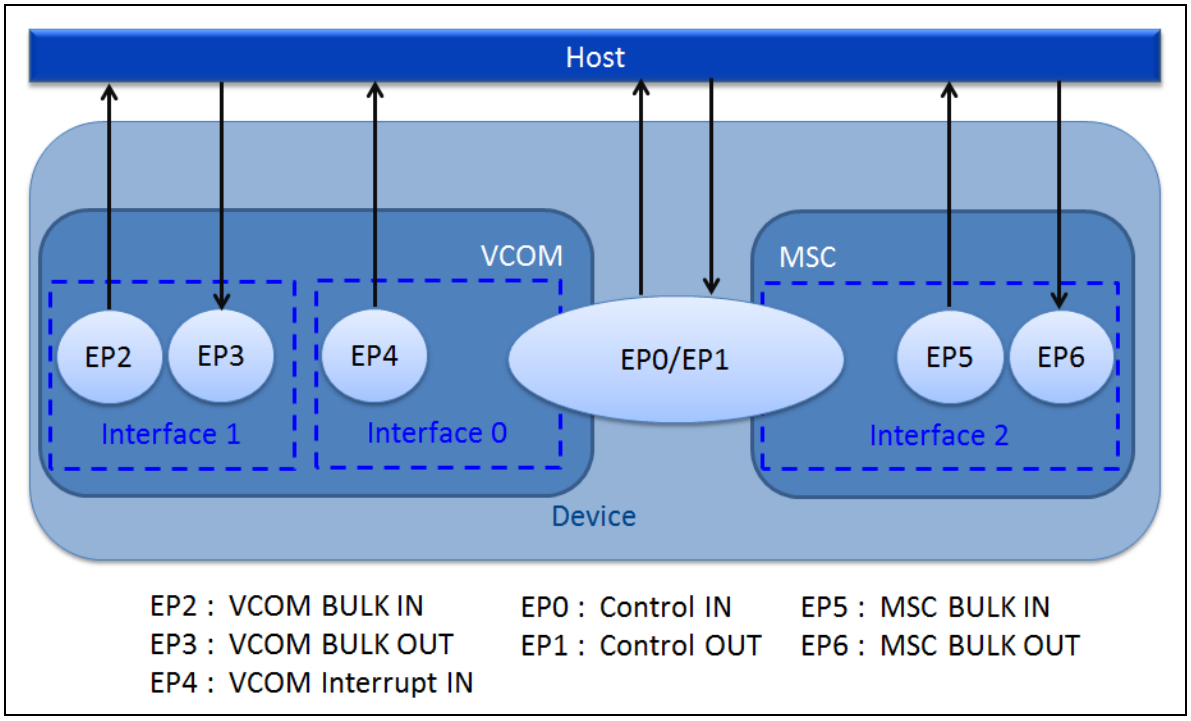


圖 18 VCOM + MSC複合裝置的架構

開發步驟如下：

步驟1：根據裝置的類別來配置端點和其對應的IN/OUT Buffer，此範例配置七個端點EP0~EP6。EP0_BUF_BASE~EP6_BUF_BASE分別為各個端點可以使用的Buffer起始位址。EP0被配置為Control IN，EP1被配置為Control OUT，EP2和EP5被配置為BULK IN，EP3和EP6被配置為BULK OUT，EP4被配置為Interrupt IN，除了EP0、EP1及EP4的最大封包長度為8Bytes，其他端點的最大封包長度皆為64Bytes。下面的程式碼顯示Endpoint的設定。

```
/* 端點最大封包長度 */
#define EP0_MAX_PKT_SIZE    8
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    64
#define EP3_MAX_PKT_SIZE    64
#define EP4_MAX_PKT_SIZE    8
#define EP5_MAX_PKT_SIZE    64
```

```
#define EP6_MAX_PKT_SIZE    64

/* 各個端點在USB Buffer的偏移值 */
#define SETUP_BUF_BASE  0
#define SETUP_BUF_LEN   8
#define EP0_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP0_BUF_LEN      EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP1_BUF_LEN      EP1_MAX_PKT_SIZE
#define EP2_BUF_BASE    (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN      EP2_MAX_PKT_SIZE
#define EP3_BUF_BASE    (EP2_BUF_BASE + EP2_BUF_LEN)
#define EP3_BUF_LEN      EP3_MAX_PKT_SIZE
#define EP4_BUF_BASE    (EP3_BUF_BASE + EP3_BUF_LEN)
#define EP4_BUF_LEN      EP4_MAX_PKT_SIZE
#define EP5_BUF_BASE    (EP4_BUF_BASE + EP4_BUF_LEN)
#define EP5_BUF_LEN      EP5_MAX_PKT_SIZE
#define EP6_BUF_BASE    (EP5_BUF_BASE + EP5_BUF_LEN)
#define EP6_BUF_LEN      EP6_MAX_PKT_SIZE

void VCOM_Init(void)
{
    /* Init setup packet buffer */
    /* Buffer for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;

    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

    /*****************************************************/
    /* EP2 ==> Bulk IN endpoint, address 1 */
```

```
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | BULK_IN_EP_NUM);
    /* Buffer offset for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

    /* EP3 ==> Bulk Out endpoint, address 2 */
    USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_OUT | BULK_OUT_EP_NUM);
    /* Buffer offset for EP3 */
    USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);
    /* trigger receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);

    /* EP4 ==> Interrupt IN endpoint, address 3 */
    USBD_CONFIG_EP(EP4, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
    /* Buffer offset for EP4  */
    USBD_SET_EP_BUF_ADDR(EP4, EP4_BUF_BASE);

    /*****************************************************/
    /* EP5 ==> Bulk IN endpoint, address 4 */
    USBD_CONFIG_EP(EP5, USBD_CFG_EPMODE_IN | BULK_IN_EP_NUM_1);
    /* Buffer range for EP5 */
    USBD_SET_EP_BUF_ADDR(EP5, EP5_BUF_BASE);

    /* EP6 ==> Bulk Out endpoint, address 5 */
    USBD_CONFIG_EP(EP6, USBD_CFG_EPMODE_OUT | BULK_OUT_EP_NUM_1);
    /* Buffer range for EP6 */
    USBD_SET_EP_BUF_ADDR(EP6, EP6_BUF_BASE);
    /* trigger to receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP6, EP6_MAX_PKT_SIZE);
}
```

步驟2：設備、配置、介面、端點和字串描述元都包含在"descriptor.c"中，使用者在此檔案中加入VCOM和MSC這兩個類別的各種描述元。因為此範例屬於多重介面的複合裝置，需要額外使用介面關聯描述元(IAD)來定義多重介面功能，因此設備描述元中的三個欄位(bDeviceClass、bDeviceSubClass及bDeviceProtocol)的值分別為：0xEF, 0x02, 0x01。

```
/* USB 設備描述元，下面的VID和PID是很重要的，它決定USB主機要安裝哪個驅動程式 */
uint8_t gu8DeviceDescriptor[] = {
    LEN_DEVICE,         /* bLength */
    DESC_DEVICE,        /* bDescriptorType */
```

```
    0x00, 0x02,         /* bcdUSB */
    0xEF,               /* bDeviceClass: IAD*/
    0x02,               /* bDeviceSubClass */
    0x01,               /* bDeviceProtocol */
    EP0_MAX_PKT_SIZE,   /* bMaxPacketSize0 */
    /* idVendor */
    USBD_VID & 0x00FF,
    (USBD_VID & 0xFF00) >> 8,
    /* idProduct */
    USBD_PID & 0x00FF,
    (USBD_PID & 0xFF00) >> 8,
    0x00, 0x03,     /* bcdDevice */
    0x01,           /* iManufacture */
    0x02,           /* iProduct */
    0x00,           /* iSerialNumber - no serial */
    0x01            /* bNumConfigurations */
}

/* USB 組態描述元，它包含介面及端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
/* 組態描述元 */
    LEN_CONFIG,         /* bLength              */
    DESC_CONFIG,        /* bDescriptorType      */
    0x6B, 0x00,         /* wTotalLength         */
    0x03,               /* bNumInterfaces       */
    0x01,               /* bConfigurationValue  */
    0x00,               /* iConfiguration       */
    0xC0,               /* bmAttributes         */
    0x32,               /* MaxPower             */

/* 介面關聯描述元(IAD) */
    0x08,               // bLength: Interface Descriptor size
    0x0B,               // bDescriptorType: IAD
    0x00,               // bFirstInterface
    0x02,               // bInterfaceCount
    0x02,               // bFunctionClass: CDC
    0x02,               // bFunctionSubClass
    0x01,               // bFunctionProtocol
    0x02,               // iFunction
```

```
/* 介面0 : VCOM Control介面描述元 */
    LEN_INTERFACE,  /* bLength              */
    DESC_INTERFACE, /* bDescriptorType      */
    0x00,           /* bInterfaceNumber     */
    0x00,           /* bAlternateSetting    */
    0x01,           /* bNumEndpoints        */
    0x02,           /* bInterfaceClass      */
    0x02,           /* bInterfaceSubClass   */
    0x01,           /* bInterfaceProtocol   */
    0x00,           /* iInterface           */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x00,           /* Header functional descriptor subtype */
    0x10, 0x01,     /* Communication device compliant to the communication
spec. ver. 1.10 */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x01,           /* Call management functional descriptor */
    0x00,           /* BIT0: Whether device handle call management itself. */
                    /* BIT1: Whether device can send/receive call management
information over a Data Class Interface 0 */
    0x01,           /* Interface number of data class interface optionally
used for call management */

    /* Communication Class Specified INTERFACE descriptor */
    0x04,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x02,           /* Abstract control management functional descriptor
subtype */
    0x00,           /* bmCapabilities       */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* bLength              */
    0x24,           /* bDescriptorType: CS_INTERFACE descriptor type */
    0x06,           /* bDescriptorSubType   */
```

```
    0x00,           /* bMasterInterface     */
    0x01,           /* bSlaveInterface0     */

/* 端點4描述元: 中斷IN，地址為INT_IN_EP_NUM */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_INPUT | INT_IN_EP_NUM),     /* bEndpointAddress */
    EP_INT,                         /* bmAttributes     */
    EP4_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x01,                           /* bInterval        */

/* 介面1 : VCOM Data介面描述元 */
    LEN_INTERFACE,                  /* bLength              */
    DESC_INTERFACE,                 /* bDescriptorType      */
    0x01,                           /* bInterfaceNumber     */
    0x00,                           /* bAlternateSetting    */
    0x02,                           /* bNumEndpoints        */
    0x0A,                           /* bInterfaceClass      */
    0x00,                           /* bInterfaceSubClass   */
    0x00,                           /* bInterfaceProtocol   */
    0x00,                           /* iInterface           */

/* 端點2描述元: BULK IN，地址為BULK_IN_EP_NUM */
    /* ENDPOINT descriptor */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_INPUT | BULK_IN_EP_NUM),    /* bEndpointAddress */
    EP_BULK,                        /* bmAttributes     */
    EP2_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x00,                           /* bInterval        */

/* 端點3描述元: BULK OUT，地址為BULK_OUT_EP_NUM */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_OUTPUT | BULK_OUT_EP_NUM),  /* bEndpointAddress */
    EP_BULK,                        /* bmAttributes     */
    EP3_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x00,                           /* bInterval        */
```

```
/* 介面2：MSC 介面描述元 */
    LEN_INTERFACE,                                  // bLength
    DESC_INTERFACE,                                 // bDescriptorType
    0x02,                                           // bInterfaceNumber
    0x00,                                           // bAlternateSetting
    0x02,                                           // bNumEndpoints
    0x08,                                           // bInterfaceClass
    0x06,                                           // bInterfaceSubClass
    0x50,                                           // bInterfaceProtocol
    0x00,                                           // iInterface

/* 端點5描述元: BULK IN，地址為BULK_IN_EP_NUM_1 */
    LEN_ENDPOINT,                       // bLength
    DESC_ENDPOINT,                      // bDescriptorType
    (EP_INPUT | BULK_IN_EP_NUM_1),      // bEndpointAddress
    EP_BULK,                            // bmAttributes
    EP5_MAX_PKT_SIZE, 0x00,             // wMaxPacketSize
    0x00,                               // bInterval

/* 端點6描述元: BULK OUT，地址為BULK_OUT_EP_NUM_1 */
    LEN_ENDPOINT,                       // bLength
    DESC_ENDPOINT,                      // bDescriptorType
    (EP_OUTPUT | BULK_OUT_EP_NUM_1), // bEndpointAddress
    EP_BULK,                            // bmAttributes
    EP6_MAX_PKT_SIZE, 0x00,             // wMaxPacketSize
    0x00,                               // bInterval
};

/* 字串描述元 */
uint8_t gu8StringLang[4] = {
    4,                                              /* bLength */
    DESC_STRING,                                    /* bDescriptorType */
    0x09, 0x04
};

/* 字串描述元 */
uint8_t gu8VendorStringDesc[] = {
    16,
    DESC_STRING,
```

```
    'N', 0, 'u', 0, 'v', 0, 'o', 0, 't', 0, 'o', 0, 'n', 0
};

/* 字串描述元 */
uint8_t gu8ProductStringDesc[] = {
    22,             /* bLength          */
    DESC_STRING,    /* bDescriptorType  */
    'U', 0, 'S', 0, 'B', 0, ' ', 0, 'D', 0, 'e', 0, 'v', 0, 'i', 0, 'c', 0,
'e', 0
};

uint8_t *gu8UsbHidReport[3] = {
    NULL,
    NULL,
    NULL,
};

uint32_t gu32UsbHidReportLen[3] = {
    0,
    0,
    0,
};

uint32_t gu32ConfigHidDescIdx[3] = {
    0,
    0,
    0,
};
```

步驟3：vcom_massStorage.c執行VCOM/MSC類別請求和端點資料處理，在USB通信設備類別（CDC）的抽象控制模式中定義虛擬COM埠設備必要的特殊類別請求有三個，MSC設備定義兩個類別請求，但是並不是所有的命令都必須回應。vcom_massStorage.c執行VCOM/MSC類別請求及端點EP2~EP6的Callback如下所示：

```
void HID_ClassRequest(void)
{
    uint8_t buf[8];
    USBD_GetSetupPacket(buf);
```

```
    if (buf[0] & 0x80) {                /* 該命令數據傳輸方向是Device -> Host */
        switch (buf[1]) {
         case GET_LINE_CODE: {  /* 介面0: VCOM的類別請求 */
                if (buf[4] == 0) { /* VCOM-1 */
                    USBD_MemCopy((uint8_t *)(USBD_BUF_BASE +
USBD_GET_EP_BUF_ADDR(EP0)), (uint8_t *)&gLineCoding, 7);
                }
                /* Data stage */
                USBD_SET_DATA1(EP0);
                USBD_SET_PAYLOAD_LEN(EP0, 7);
                /* Status stage */
                USBD_PrepareCtrlOut(0,0);
                break;
            }
        case GET_MAX_LUN: {             /* 介面2: MSC的類別請求 */
            if ((((buf[3]<<8)+buf[2]) == 0) && (((buf[5]<<8)+buf[4]) == 0) &&
(((buf[7]<<8)+buf[6]) == 1)) {
                M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) = 0;
                /* Data stage */
                USBD_SET_DATA1(EP0);
                USBD_SET_PAYLOAD_LEN(EP0, 1);
                /* Status stage */
                USBD_PrepareCtrlOut(0,0);
            }
            else {/* Invalid Get MaxLun command */
                USBD_SetStall(0);
            }
            break;
        }
    }
    else {
    /* Host -> Device */
    switch (buf[1]) {
       case SET_CONTROL_LINE_STATE: {           /* 介面0: VCOM的類別請求 */
            if (buf[4] == 0) { /* VCOM-1 */
                gCtrlSignal = buf[3];
                gCtrlSignal = (gCtrlSignal << 8) | buf[2];
            }
```

```
            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;
        }
        case SET_LINE_CODE: {                   /* 介面0: VCOM的類別請求 */
            if (buf[4] == 0) /* VCOM-1 */
                USBD_PrepareCtrlOut((uint8_t *)&gLineCoding, 7);

            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);

            break;
        }
        case BULK_ONLY_MASS_STORAGE_RESET: {         /* 介面2: MSC的類別請求 */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            USBD_LockEpStall(0);

            /* Clear ready */
            USBD->EP[EP5].CFG |= USBD_CFG_CLRRDY_Msk;
            USBD->EP[EP6].CFG |= USBD_CFG_CLRRDY_Msk;

            /* Prepare to receive the CBW */
            g_u8EP5Ready = 0;
            g_u8BulkState = BULK_CBW;

            USBD_SET_DATA1(EP6);
            USBD_SET_EP_BUF_ADDR(EP6, g_u32BulkBuf0);
            USBD_SET_PAYLOAD_LEN(EP6, EP6_MAX_PKT_SIZE);
            break;
        }
        default: {
            // Stall
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
```

```
        }
    }
}
void EP2_Handler(void)                                  /* BULK IN handler */
{
    gu32TxSize = 0;
}

void EP3_Handler(void)                                 /* BULK OUT handler */
{
    gu32RxSize = USBD_GET_PAYLOAD_LEN(EP3);
    gpu8RxBuf = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP3));

    /* Set a flag to indicate bulk out ready */
    gi8BulkOutReady = 1;
}

void EP5_Handler(void)                                  /* BULK IN handler */
{
    g_u8EP5Ready = 1;
    MSC_AckCmd();
}

void EP6_Handler(void)                                 /* BULK OUT handler */
{
    g_u8EP6Ready = 1;
}
```

在第一次執行此範例與電腦連接時，需要先安裝驅動程式，此時搜尋此範例搭配的INF檔案進行安裝。安裝完成後，在電腦的裝置管理員中可以看到新增的Nuvoton USB虛擬COM埠，如圖 19所示。

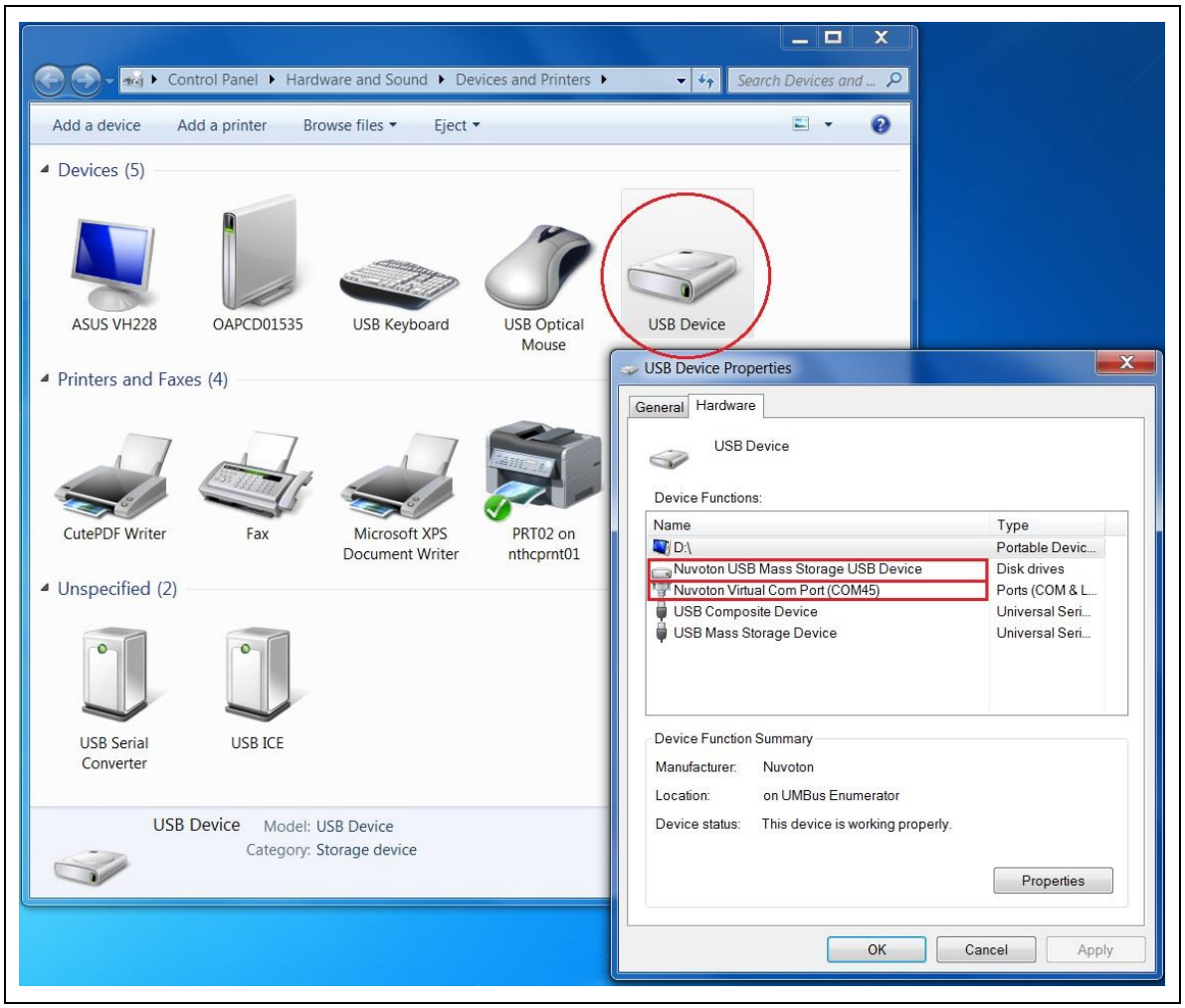


圖 19 VCOM + MSC 複合裝置

3.3.6 VCOM + HID Keyboard

這個複合裝置是由VCOM和HID Keyboard組成。圖 20說明VCOM + HID Keyboard複合裝置範例的架構。首先可以先從BSP\SampleCode\StdDriver分別取得VCOM和HID Keyboard這兩個獨立功能的範例程式，使用者可以藉由使用VCOM的範例程式並結合HID Keyboard的範例程式來實作出新的VCOM和HID Keyboard的複合裝置範例。

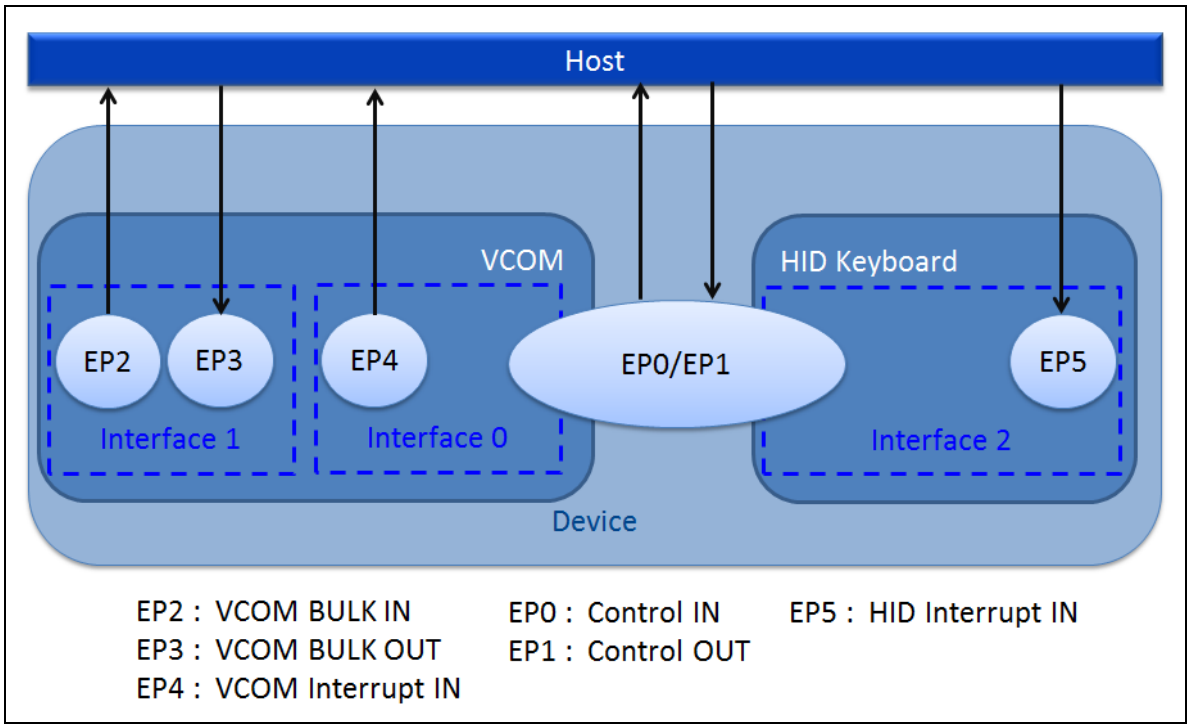


圖 20 VCOM + HID Keyboard複合裝置的架構

開發步驟如下：

步驟1：根據裝置的類別來配置端點和其對應的IN/OUT Buffer，此範例配置六個端點EP0~EP5。EP0_BUF_BASE~EP5_BUF_BASE分別為各個端點可以使用的Buffer起始位址。EP0被配置為Control IN，EP1被配置為Control OUT，EP2被配置為BULK IN，EP3被配置為BULK OUT，EP4和EP5被配置為Interrupt IN，除了EP2及EP3的最大封包長度為64Bytes，其他端點的最大封包長度皆為8Bytes。下面的程式碼顯示Endpoint的設定。

```
/* 端點最大封包長度 */
#define EP0_MAX_PKT_SIZE    8
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    64
#define EP3_MAX_PKT_SIZE    64
#define EP4_MAX_PKT_SIZE    8
#define EP5_MAX_PKT_SIZE    8

/* 各個端點在USB Buffer的偏移值 */
#define SETUP_BUF_BASE  0
#define SETUP_BUF_LEN   8
#define EP0_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
```

```
#define EP0_BUF_LEN       EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP1_BUF_LEN       EP1_MAX_PKT_SIZE
#define EP2_BUF_BASE    (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN       EP2_MAX_PKT_SIZE
#define EP3_BUF_BASE    (EP2_BUF_BASE + EP2_BUF_LEN)
#define EP3_BUF_LEN       EP3_MAX_PKT_SIZE
#define EP4_BUF_BASE    (EP3_BUF_BASE + EP3_BUF_LEN)
#define EP4_BUF_LEN       EP4_MAX_PKT_SIZE
#define EP5_BUF_BASE    (EP4_BUF_BASE + EP4_BUF_LEN)
#define EP5_BUF_LEN       EP5_MAX_PKT_SIZE

void HID_Init(void)
{
    /* Init setup packet buffer */
    /* Buffer range for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;

    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

        /*****************************************************/
    /* EP2 ==> Bulk IN endpoint, address 1 */
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | BULK_IN_EP_NUM);
    /* Buffer offset for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

    /* EP3 ==> Bulk Out endpoint, address 2 */
    USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_OUT | BULK_OUT_EP_NUM);
    /* Buffer offset for EP3 */
    USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);
```

```
    /* trigger receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);

    /* EP4 ==> Interrupt IN endpoint, address 3 */
    USBD_CONFIG_EP(EP4, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
    /* Buffer offset for EP4  */
    USBD_SET_EP_BUF_ADDR(EP4, EP4_BUF_BASE);

    /*****************************************************/
    /* EP5 ==> Interrupt IN endpoint, address 4 */
    USBD_CONFIG_EP(EP5, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM_1);
    /* Buffer range for EP5 */
    USBD_SET_EP_BUF_ADDR(EP5, EP5_BUF_BASE);
}
```

步驟2：設備、配置、介面、端點和字串描述元都包含在"descriptor.c"中，使用者在此檔案中加入VCOM和HID Transfer這兩個類別的各種描述元。因為此範例屬於多重介面的複合裝置，需要額外使用介面關聯描述元(IAD)來定義多重介面功能，因此設備描述元中的三個欄位(bDeviceClass、bDeviceSubClass及bDeviceProtocol)的值分別為：0xEF, 0x02, 0x01。

```
/* USB設備描述元， VID和PID決定USB主機要安裝哪個驅動程式 */
uint8_t gu8DeviceDescriptor[] = {
    LEN_DEVICE,         /* bLength */
    DESC_DEVICE,        /* bDescriptorType */
    0x00, 0x02,         /* bcdUSB */
    0xEF,               /* bDeviceClass: IAD*/
    0x02,               /* bDeviceSubClass */
    0x01,               /* bDeviceProtocol */
    EP0_MAX_PKT_SIZE,   /* bMaxPacketSize0 */
    /* idVendor */
    USBD_VID & 0x00FF,
    (USBD_VID & 0xFF00) >> 8,
    /* idProduct */
    USBD_PID & 0x00FF,
    (USBD_PID & 0xFF00) >> 8,
    0x00, 0x03,     /* bcdDevice */
    0x01,           /* iManufacture */
    0x02,           /* iProduct */
    0x00,           /* iSerialNumber - no serial */
```

```
    0x01            /* bNumConfigurations */
}

/* USB 組態描述元，它包含介面、HID及端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
/* 組態描述元 */
    LEN_CONFIG,         /* bLength              */
    DESC_CONFIG,        /* bDescriptorType      */
    0x6B, 0x00,         /* wTotalLength         */
    0x03,               /* bNumInterfaces       */
    0x01,               /* bConfigurationValue  */
    0x00,               /* iConfiguration       */
    0xC0,               /* bmAttributes         */
    0x32,               /* MaxPower             */

/* 介面關聯描述元(IAD) */
    0x08,               // bLength: Interface Descriptor size
    0x0B,               // bDescriptorType: IAD
    0x00,               // bFirstInterface
    0x02,               // bInterfaceCount
    0x02,               // bFunctionClass: CDC
    0x02,               // bFunctionSubClass
    0x01,               // bFunctionProtocol
    0x02,               // iFunction

    LEN_INTERFACE,  /* bLength              */
    DESC_INTERFACE, /* bDescriptorType      */
    0x00,           /* bInterfaceNumber     */
    0x00,           /* bAlternateSetting    */
    0x01,           /* bNumEndpoints        */
    0x02,           /* bInterfaceClass      */
    0x02,           /* bInterfaceSubClass   */
    0x01,           /* bInterfaceProtocol   */
    0x00,           /* iInterface           */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x00,           /* Header functional descriptor subtype */
```

```
    0x10, 0x01,     /* Communication device compliant to the communication
spec. ver. 1.10 */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x01,           /* Call management functional descriptor */
    0x00,           /* BIT0: Whether device handle call management itself. */
                    /* BIT1: Whether device can send/receive call management
information over a Data Class Interface 0 */
    0x01,           /* Interface number of data class interface optionally
used for call management */

    /* Communication Class Specified INTERFACE descriptor */
    0x04,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x02,           /* Abstract control management functional descriptor
subtype */
    0x00,           /* bmCapabilities       */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* bLength              */
    0x24,           /* bDescriptorType: CS_INTERFACE descriptor type */
    0x06,           /* bDescriptorSubType   */
    0x00,           /* bMasterInterface     */
    0x01,           /* bSlaveInterface0     */

/* 端點4描述元: 中斷IN，地址為INT_IN_EP_NUM */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_INPUT | INT_IN_EP_NUM),     /* bEndpointAddress */
    EP_INT,                         /* bmAttributes     */
    EP4_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x01,                           /* bInterval        */

/* 介面1 : VCOM Data介面描述元 */
    LEN_INTERFACE,                  /* bLength              */
    DESC_INTERFACE,                 /* bDescriptorType      */
    0x01,                           /* bInterfaceNumber     */
    0x00,                           /* bAlternateSetting    */
```

```
    0x02,                           /* bNumEndpoints        */
    0x0A,                           /* bInterfaceClass      */
    0x00,                           /* bInterfaceSubClass   */
    0x00,                           /* bInterfaceProtocol   */
    0x00,                           /* iInterface           */

/* 端點2描述元: BULK IN，地址為BULK_IN_EP_NUM */
    /* ENDPOINT descriptor */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_INPUT | BULK_IN_EP_NUM),    /* bEndpointAddress */
    EP_BULK,                        /* bmAttributes     */
    EP2_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x00,                           /* bInterval        */

/* 端點3描述元: BULK OUT，地址為BULK_OUT_EP_NUM */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_OUTPUT | BULK_OUT_EP_NUM),  /* bEndpointAddress */
    EP_BULK,                        /* bmAttributes     */
    EP3_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x00,                           /* bInterval        */

/* 介面2 : HID Keyboard介面描述元 */
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
    0x02,           /* bInterfaceNumber */
    0x00,           /* bAlternateSetting */
    0x01,           /* bNumEndpoints */
    0x03,           /* bInterfaceClass */
    0x00,           /* bInterfaceSubClass */
    HID_KEYBOARD,   /* bInterfaceProtocol */
    0x00,           /* iInterface */

/* HID描述元 */
    LEN_HID,        /* Size of this descriptor in UINT8s. */
    DESC_HID,       /* HID descriptor type. */
    0x10, 0x01,     /* HID Class Spec. release number. */
    0x00,           /* H/W target country. */
```

```
    0x01,           /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,   /* Descriptor type. */
    /* Total length of report descriptor. */
    sizeof(HID_KeyboardReportDescriptor) & 0x00FF,
    (sizeof(HID_KeyboardReportDescriptor) & 0xFF00) >> 8,

/* 端點5描述元: 中斷IN，地址為INT_IN_EP_NUM_1 */
    LEN_ENDPOINT,   /* bLength */
    DESC_ENDPOINT,  /* bDescriptorType */
    (INT_IN_EP_NUM_1 | EP_INPUT), /* bEndpointAddress */
    EP_INT,         /* bmAttributes */
    /* wMaxPacketSize */
    EP5_MAX_PKT_SIZE & 0x00FF,
    (EP5_MAX_PKT_SIZE & 0xFF00) >> 8,
    HID_DEFAULT_INT_IN_INTERVAL     /* bInterval */
};

/* HID Keyboard報告描述元 */
uint8_t HID_KeyboardReportDescriptor[] =
{
    0x05, 0x01,         /* Usage Page(Generic Desktop Controls) */
    0x09, 0x06,         /* Usage(Keyboard) */
    0xA1, 0x01,         /* Collection(Application) */
    0x05, 0x07,         /* Usage Page(Keyboard/Keypad) */
    0x19, 0xE0,         /* Usage Minimum(0xE0) */
    0x29, 0xE7,         /* Usage Maximum(0xE7) */
    0x15, 0x00,         /* Logical Minimum(0x0) */
    0x25, 0x01,         /* Logical Maximum(0x1) */
    0x75, 0x01,         /* Report Size(0x1) */
    0x95, 0x08,         /* Report Count(0x8) */
    0x81, 0x02,         /* Input (Data) => Modifier byte */
    0x95, 0x01,         /* Report Count(0x1) */
    0x75, 0x08,         /* Report Size(0x8) */
    0x81, 0x01,         /* Input (Constant) => Reserved byte */
    0x95, 0x05,         /* Report Count(0x5) */
    0x75, 0x01,         /* Report Size(0x1) */
    0x05, 0x08,         /* Usage Page(LEDs) */
    0x19, 0x01,         /* Usage Minimum(0x1) */
    0x29, 0x05,         /* Usage Maximum(0x5) */
```

```
    0x91, 0x02,         /* Output (Data) => LED report */
    0x95, 0x01,         /* Report Count(0x1) */
    0x75, 0x03,         /* Report Size(0x3) */
    0x91, 0x01,         /* Output (Constant) => LED report padding */
    0x95, 0x06,         /* Report Count(0x6) */
    0x75, 0x08,         /* Report Size(0x8) */
    0x15, 0x00,         /* Logical Minimum(0x0) */
    0x25, 0x65,         /* Logical Maximum(0x65) */
    0x05, 0x07,         /* Usage Page(Keyboard/Keypad) */
    0x19, 0x00,         /* Usage Minimum(0x0) */
    0x29, 0x65,         /* Usage Maximum(0x65) */
    0x81, 0x00,         /* Input (Data) */
    0xC0,               /* End Collection */
};

/* 字串描述元 */
uint8_t gu8StringLang[4] = {
    4,                                          /* bLength */
    DESC_STRING,                                /* bDescriptorType */
    0x09, 0x04
};

/* 字串描述元 */
uint8_t gu8VendorStringDesc[] = {
    16,
    DESC_STRING,
    'N', 0, 'u', 0, 'v', 0, 'o', 0, 't', 0, 'o', 0, 'n', 0
};

/* 字串描述元 */
uint8_t gu8ProductStringDesc[] = {
    22,             /* bLength          */
    DESC_STRING,    /* bDescriptorType  */
    'U', 0, 'S', 0, 'B', 0, ' ', 0, 'D', 0, 'e', 0, 'v', 0, 'i', 0, 'c', 0,
'e', 0
};

/* 因為此範例的HID為介面2，因此HID相關結構的Index皆為2 */
uint8_t *gu8UsbHidReport[3] = {
```

```
    NULL,
    NULL,
    HID_KeyboardReportDescriptor,             /* HID Keyboard報告描述元 */
};

uint32_t gu32UsbHidReportLen[3] = {
    0,
    0,
    sizeof(HID_KeyboardReportDescriptor),     /* HID Keyboard報告描述元的長度 */
};

uint32_t gu32ConfigHidDescIdx[3] = {
    0,
    0,
/* HID Keyboard描述元在組態描述元內的位置 */
    (sizeof(gu8ConfigDescriptor) - LEN_HID - LEN_ENDPOINT)
};
```

步驟3：VCOM_And_hid_transfer.c執行VCOM/HID類別請求和端點資料處理，在USB通信設備類別（CDC）的抽象控制模式中定義虛擬COM埠設備必要的特殊類別請求有三個，HID設備定義六個類別請求，但是並不是所有的命令都必須回應。VCOM_And_hid_transfer.c執行VCOM/HID類別請求及端點EP2~EP6的Callback如下所示：

```
void HID_ClassRequest(void)
{
    uint8_t buf[8];
    USBD_GetSetupPacket(buf);

    if (buf[0] & 0x80) {                /* 該命令數據傳輸方向是Device -> Host */
        switch (buf[1]) {
         case GET_LINE_CODE: {  /* 介面0: VCOM的類別請求 */
                if (buf[4] == 0) { /* VCOM-1 */
                    USBD_MemCopy((uint8_t *)(USBD_BUF_BASE +
USBD_GET_EP_BUF_ADDR(EP0)), (uint8_t *)&gLineCoding, 7);
                }
                /* Data stage */
                USBD_SET_DATA1(EP0);
                USBD_SET_PAYLOAD_LEN(EP0, 7);
                /* Status stage */
```

```
                USBD_PrepareCtrlOut(0,0);
                break;
            }
        /* 介面2: HID的類別請求 */
        case GET_REPORT:                                        /* 不處理 */
        case GET_IDLE:                                          /* 不處理 */
        case GET_PROTOCOL:                                      /* 不處理 */
        default: {
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
        }
    }
    else {
    /* Host -> Device */
    switch (buf[1]) {
       case SET_CONTROL_LINE_STATE: {           /* 介面0: VCOM的類別請求 */
            if (buf[4] == 0) { /* VCOM-1 */
                gCtrlSignal = buf[3];
                gCtrlSignal = (gCtrlSignal << 8) | buf[2];
            }

            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;
        }
        case SET_LINE_CODE: {                   /* 介面0: VCOM的類別請求 */
            if (buf[4] == 0) /* VCOM-1 */
                USBD_PrepareCtrlOut((uint8_t *)&gLineCoding, 7);

            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);

            break;
        }
        /* 介面2: HID的類別請求 */
```

```
        case SET_REPORT: {
            if (buf[3] == 3) {
                /* Request Type = Feature */
                USBD_SET_DATA1(EP1);
                USBD_SET_PAYLOAD_LEN(EP1, 0);
            }
            break;
        }
        /* 介面2: HID的類別請求 */
        case SET_IDLE: {
            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;
        }
        /* 介面2: HID的類別請求 */
        case SET_PROTOCOL:                                          /* 不處理 */
        default: {
            // Stall
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
        }
    }
}

void EP2_Handler(void)  /* BULK IN handler */
{
    gu32TxSize = 0;
}

void EP3_Handler(void)  /* BULK OUT handler */
{
    /* Bulk OUT */
    gu32RxSize = USBD_GET_PAYLOAD_LEN(EP3);
    gpu8RxBuf = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP3));

    /* Set a flag to indicate bulk out ready */
```

```
    gi8BulkOutReady = 1;
}

void EP5_Handler(void)  /* Interrupt IN handler */
{
    g_u8EP5Ready = 1;
}
```

在第一次執行此範例與PC連接時需要安裝驅動程式，此時搜尋此範例搭配的INF檔案進行安裝。安裝完成後，在電腦的裝置管理員中可以看到新增的Nuvoton USB虛擬COM埠，圖 21所示。

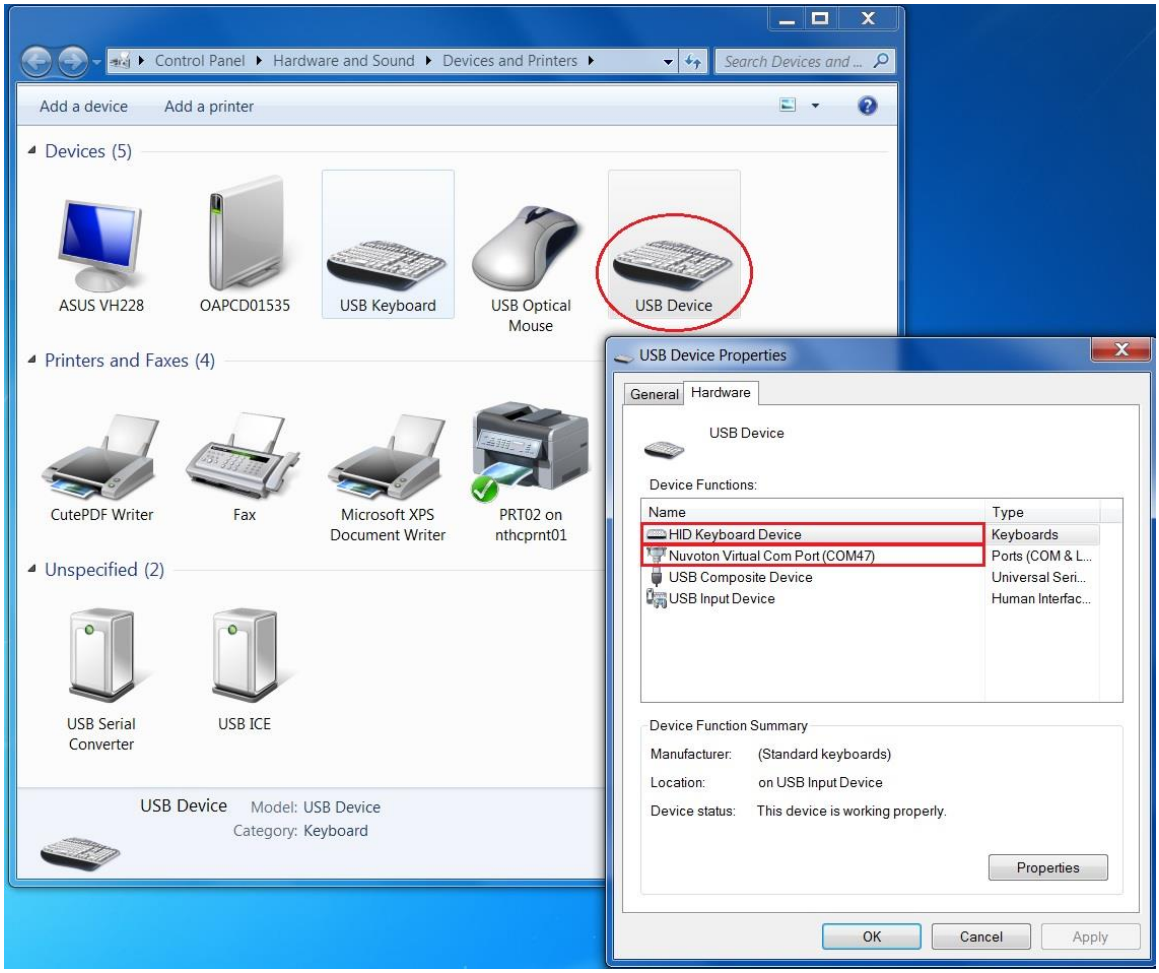


圖 21 VCOM + HID Keyboard複合裝置

3.3.7 Dual VCOM

此範例是結合兩個VCOM port的複合裝置。圖 22說明Dual VCOM複合裝置範

例的架構。首先可以先從BSP\SampleCode\StdDriver取得VCOM的範例程式，使用者可以藉由使用VCOM的範例程式實作出新的兩個VCOM的複合裝置範例。

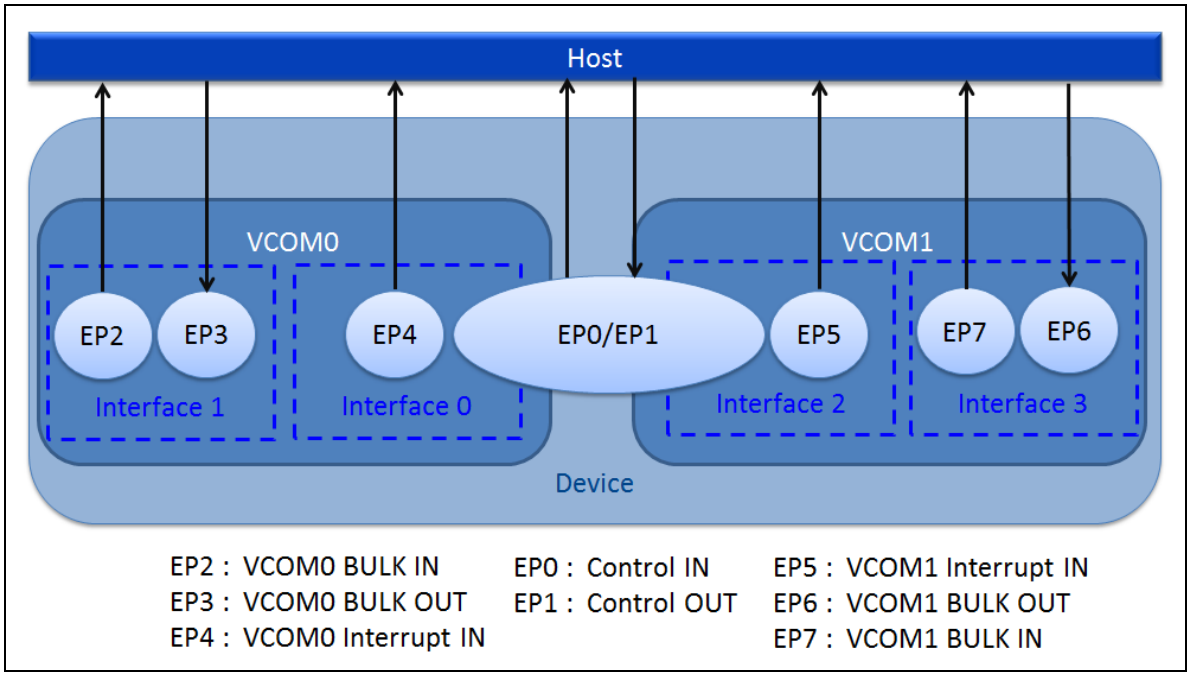


圖 22 Dual VCOM複合裝置的架構

開發步驟如下：

步驟1：根據裝置的類別來配置端點和其對應的IN/OUT Buffer，此範例配置八個端點EP0~EP7。EP0_BUF_BASE~EP7_BUF_BASE分別為各個端點可以使用的Buffer起始位址。EP0被配置為Control IN，EP1被配置為Control OUT，EP2和EP7被配置為BULK IN，EP3和EP6被配置為BULK OUT，EP4和EP5被配置為Interrupt IN，除了EP0、EP1、EP4及EP5的最大封包長度為8Bytes，其他端點的最大封包長度皆為64Bytes。下面的程式碼顯示Endpoint的設定。

```
/* 端點最大封包長度 */
#define EP0_MAX_PKT_SIZE    64
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    64
#define EP3_MAX_PKT_SIZE    64
#define EP4_MAX_PKT_SIZE    8
#define EP5_MAX_PKT_SIZE    8
#define EP6_MAX_PKT_SIZE    64
#define EP7_MAX_PKT_SIZE    64

/* 各個端點在USB Buffer的偏移值 */
```

```
#define SETUP_BUF_BASE  0
#define SETUP_BUF_LEN   8
#define EP0_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP0_BUF_LEN      EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP1_BUF_LEN      EP1_MAX_PKT_SIZE
#define EP2_BUF_BASE    (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN      EP2_MAX_PKT_SIZE
#define EP3_BUF_BASE    (EP2_BUF_BASE + EP2_BUF_LEN)
#define EP3_BUF_LEN      EP3_MAX_PKT_SIZE
#define EP4_BUF_BASE    (EP3_BUF_BASE + EP3_BUF_LEN)
#define EP4_BUF_LEN      EP4_MAX_PKT_SIZE
#define EP5_BUF_BASE    (EP4_BUF_BASE + EP4_BUF_LEN)
#define EP5_BUF_LEN      EP5_MAX_PKT_SIZE
#define EP6_BUF_BASE    (EP5_BUF_BASE + EP5_BUF_LEN)
#define EP6_BUF_LEN      EP6_MAX_PKT_SIZE
#define EP7_BUF_BASE    (EP6_BUF_BASE + EP6_BUF_LEN)
#define EP7_BUF_LEN      EP7_MAX_PKT_SIZE

void VCOM_Init(void)
{
    /* Init setup packet buffer */
    /* Buffer for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;

    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

    /*****************************************************/
    /* EP2 ==> Bulk IN endpoint, address 1 */
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | BULK_IN_EP_NUM);
```

```
    /* Buffer offset for EP2 */
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

    /* EP3 ==> Bulk Out endpoint, address 2 */
    USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_OUT | BULK_OUT_EP_NUM);
    /* Buffer offset for EP3 */
    USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);
    /* trigger receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);

    /* EP4 ==> Interrupt IN endpoint, address 3 */
    USBD_CONFIG_EP(EP4, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
    /* Buffer offset for EP4 */
    USBD_SET_EP_BUF_ADDR(EP4, EP4_BUF_BASE);

    /*****************************************************/
    /* EP5 ==> Interrupt IN endpoint, address 6 */
    USBD_CONFIG_EP(EP5, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM_1);
    /* Buffer offset for EP5 */
    USBD_SET_EP_BUF_ADDR(EP5, EP5_BUF_BASE);

    /* EP6 ==> Bulk Out endpoint, address 5 */
    USBD_CONFIG_EP(EP6, USBD_CFG_EPMODE_OUT | BULK_OUT_EP_NUM_1);
    /* Buffer offset for EP6 */
    USBD_SET_EP_BUF_ADDR(EP6, EP6_BUF_BASE);
    /* trigger receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP6, EP6_MAX_PKT_SIZE);

    /* EP7 ==> Bulk IN endpoint, address 4 */
    USBD_CONFIG_EP(EP7, USBD_CFG_EPMODE_IN | BULK_IN_EP_NUM_1);
    /* Buffer offset for EP7 */
    USBD_SET_EP_BUF_ADDR(EP7, EP7_BUF_BASE);
}
```

步驟2：設備、配置、介面、端點和字串描述元都包含在"descriptor.c"中，使用者在此檔案中加入VCOM的各種描述元。因為此範例屬於多重介面的複合裝置，需要額外使用介面關聯描述元(IAD)來定義多重介面功能，因此設備描述元中的三個欄位(bDeviceClass、bDeviceSubClass及bDeviceProtocol)的值分別為：0xEF, 0x02, 0x01。

```
/* USB設備描述元， VID和PID決定USB主機要安裝哪個驅動程式 */
uint8_t gu8DeviceDescriptor[] = {
    LEN_DEVICE,         /* bLength */
    DESC_DEVICE,        /* bDescriptorType */
    0x00, 0x02,         /* bcdUSB */
    0xEF,               /* bDeviceClass: IAD*/
    0x02,               /* bDeviceSubClass */
    0x01,               /* bDeviceProtocol */
    EP0_MAX_PKT_SIZE,   /* bMaxPacketSize0 */
    /* idVendor */
    USBD_VID & 0x00FF,
    (USBD_VID & 0xFF00) >> 8,
    /* idProduct */
    USBD_PID & 0x00FF,
    (USBD_PID & 0xFF00) >> 8,
    0x00, 0x03,     /* bcdDevice */
    0x01,           /* iManufacture */
    0x02,           /* iProduct */
    0x00,           /* iSerialNumber - no serial */
    0x01            /* bNumConfigurations */
}

/* USB 組態描述元，它包含介面及端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
/* 組態描述元 */
    LEN_CONFIG,     /* bLength              */
    DESC_CONFIG,    /* bDescriptorType      */
    0x8D, 0x00,     /* wTotalLength         */
    0x04,           /* bNumInterfaces       */
    0x01,           /* bConfigurationValue  */
    0x00,           /* iConfiguration       */
    0xC0,           /* bmAttributes         */
    0x32,           /* MaxPower             */

/* 介面關聯描述元(IAD) */
    0x08,               // bLength: Interface Descriptor size
    0x0B,               // bDescriptorType: IAD
    0x00,               // bFirstInterface
    0x02,               // bInterfaceCount
```

```
    0x02,               // bFunctionClass: CDC
    0x02,               // bFunctionSubClass
    0x01,               // bFunctionProtocol
    0x02,               // iFunction

/* 介面0 : VCOM0 Control介面描述元 */
    LEN_INTERFACE,  /* bLength              */
    DESC_INTERFACE, /* bDescriptorType      */
    0x00,           /* bInterfaceNumber     */
    0x00,           /* bAlternateSetting    */
    0x01,           /* bNumEndpoints        */
    0x02,           /* bInterfaceClass      */
    0x02,           /* bInterfaceSubClass   */
    0x01,           /* bInterfaceProtocol   */
    0x00,           /* iInterface           */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x00,           /* Header functional descriptor subtype */
    0x10, 0x01,     /* Communication device compliant to the communication
spec. ver. 1.10 */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x01,           /* Call management functional descriptor */
    0x00,           /* BIT0: Whether device handle call management itself. */
                    /* BIT1: Whether device can send/receive call management
information over a Data Class Interface 0 */
    0x01,           /* Interface number of data class interface optionally
used for call management */

    /* Communication Class Specified INTERFACE descriptor */
    0x04,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x02,           /* Abstract control management functional descriptor
subtype */
    0x00,           /* bmCapabilities       */
```

```
    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* bLength              */
    0x24,           /* bDescriptorType: CS_INTERFACE descriptor type */
    0x06,           /* bDescriptorSubType   */
    0x00,           /* bMasterInterface     */
    0x01,           /* bSlaveInterface0     */

/* 端點4描述元: 中斷IN，地址為INT_IN_EP_NUM */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_INPUT | INT_IN_EP_NUM),     /* bEndpointAddress */
    EP_INT,                         /* bmAttributes     */
    EP4_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x01,                           /* bInterval        */

/* 介面1 : VCOM Data介面描述元 */
    LEN_INTERFACE,                  /* bLength              */
    DESC_INTERFACE,                 /* bDescriptorType      */
    0x01,                           /* bInterfaceNumber     */
    0x00,                           /* bAlternateSetting    */
    0x02,                           /* bNumEndpoints        */
    0x0A,                           /* bInterfaceClass      */
    0x00,                           /* bInterfaceSubClass   */
    0x00,                           /* bInterfaceProtocol   */
    0x00,                           /* iInterface           */

/* 端點2描述元: BULK IN，地址為BULK_IN_EP_NUM */
    /* ENDPOINT descriptor */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_INPUT | BULK_IN_EP_NUM),    /* bEndpointAddress */
    EP_BULK,                        /* bmAttributes     */
    EP2_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x00,                           /* bInterval        */

/* 端點3描述元: BULK OUT，地址為BULK_OUT_EP_NUM */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_OUTPUT | BULK_OUT_EP_NUM),  /* bEndpointAddress */
```

```
    EP_BULK,                        /* bmAttributes     */
    EP3_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x00,                           /* bInterval        */

/* 介面關聯描述元(IAD) */
    0x08,               // bLength: Interface Descriptor size
    0x0B,               // bDescriptorType: IAD
    0x02,               // bFirstInterface
    0x02,               // bInterfaceCount
    0x02,               // bFunctionClass: CDC
    0x02,               // bFunctionSubClass
    0x01,               // bFunctionProtocol
    0x02,               // iFunction

/* 介面2 : VCOM1 Control介面描述元 */
    LEN_INTERFACE,  /* bLength              */
    DESC_INTERFACE, /* bDescriptorType      */
    0x02,           /* bInterfaceNumber     */
    0x00,           /* bAlternateSetting    */
    0x01,           /* bNumEndpoints        */
    0x02,           /* bInterfaceClass      */
    0x02,           /* bInterfaceSubClass   */
    0x01,           /* bInterfaceProtocol   */
    0x00,           /* iInterface           */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x00,           /* Header functional descriptor subtype */
    0x10, 0x01,     /* Communication device compliant to the communication
spec. ver. 1.10 */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x01,           /* Call management functional descriptor */
    0x00,           /* BIT0: Whether device handle call management itself. */
                    /* BIT1: Whether device can send/receive call management
information over a Data Class Interface 0 */
```

```
    0x01,           /* Interface number of data class interface optionally
used for call management */

    /* Communication Class Specified INTERFACE descriptor */
    0x04,           /* Size of the descriptor, in bytes */
    0x24,           /* CS_INTERFACE descriptor type */
    0x02,           /* Abstract control management functional descriptor
subtype */
    0x00,           /* bmCapabilities       */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,           /* bLength              */
    0x24,           /* bDescriptorType: CS_INTERFACE descriptor type */
    0x06,           /* bDescriptorSubType   */
    0x00,           /* bMasterInterface     */
    0x01,           /* bSlaveInterface0     */

/* 端點5描述元: 中斷IN，地址為INT_IN_EP_NUM_1 */
    LEN_ENDPOINT,                   /* bLength          */
    DESC_ENDPOINT,                  /* bDescriptorType  */
    (EP_INPUT | INT_IN_EP_NUM_1),   /* bEndpointAddress */
    EP_INT,                         /* bmAttributes     */
    EP5_MAX_PKT_SIZE, 0x00,         /* wMaxPacketSize   */
    0x01,                           /* bInterval        */

/* 介面3 : VCOM1 Data介面描述元*/
    LEN_INTERFACE,  /* bLength              */
    DESC_INTERFACE, /* bDescriptorType      */
    0x03,           /* bInterfaceNumber     */
    0x00,           /* bAlternateSetting    */
    0x02,           /* bNumEndpoints        */
    0x0A,           /* bInterfaceClass      */
    0x00,           /* bInterfaceSubClass   */
    0x00,           /* bInterfaceProtocol   */
    0x00,           /* iInterface           */

/* 端點6描述元: BULK OUT，地址為BULK_OUT_EP_NUM_1 */
    LEN_ENDPOINT,                    /* bLength          */
    DESC_ENDPOINT,                   /* bDescriptorType  */
```

```
    (EP_OUTPUT | BULK_OUT_EP_NUM_1), /* bEndpointAddress */
    EP_BULK,                         /* bmAttributes     */
    EP6_MAX_PKT_SIZE, 0x00,          /* wMaxPacketSize   */
    0x00,                            /* bInterval        */

/* 端點7描述元: BULK IN，地址為BULK_IN_EP_NUM_1 */
    LEN_ENDPOINT,                    /* bLength          */
    DESC_ENDPOINT,                   /* bDescriptorType  */
    (EP_INPUT | BULK_IN_EP_NUM_1),   /* bEndpointAddress */
    EP_BULK,                         /* bmAttributes     */
    EP7_MAX_PKT_SIZE, 0x00,          /* wMaxPacketSize   */
    0x00,                            /* bInterval        */
};

/* 字串描述元 */
uint8_t gu8StringLang[4] = {
    4,                                          /* bLength */
    DESC_STRING,                                /* bDescriptorType */
    0x09, 0x04
};

/* 字串描述元 */
uint8_t gu8VendorStringDesc[] = {
    16,
    DESC_STRING,
    'N', 0, 'u', 0, 'v', 0, 'o', 0, 't', 0, 'o', 0, 'n', 0
};

/* 字串描述元 */
uint8_t gu8ProductStringDesc[] = {
    22,             /* bLength          */
    DESC_STRING,    /* bDescriptorType  */
    'U', 0, 'S', 0, 'B', 0, ' ', 0, 'D', 0, 'e', 0, 'v', 0, 'i', 0, 'c', 0,
'e', 0
};

uint8_t *gu8UsbHidReport[3] = {
    NULL,
    NULL,
```

```
    NULL,
};

uint32_t gu32UsbHidReportLen[3] = {
    0,
    0,
    0,
};

uint32_t gu32ConfigHidDescIdx[3] = {
    0,
    0,
    0,
};
```

步驟3：vcom_serial.c執行VCOM類別請求和端點資料處理，在USB通信設備類別（CDC）的抽象控制模式中定義虛擬COM埠設備必要的特殊類別請求有三個，但需要根據VCOM0還是VCOM1的請求來個別回應處理。vcom_serial.c執行VCOM類別請求及端點EP2~EP7的Callback如下所示：

```
void HID_ClassRequest(void)
{
    uint8_t buf[8];
    USBD_GetSetupPacket(buf);

    if (buf[0] & 0x80) {                /* 該命令數據傳輸方向是Device -> Host */
        switch (buf[1]) {
         case GET_LINE_CODE: {                              /* VCOM的類別請求 */
            if (buf[4] == 0) {  /* 介面0: VCOM0的類別請求 */
                USBD_MemCopy((uint8_t *)(USBD_BUF_BASE +
USBD_GET_EP_BUF_ADDR(EP0)), (uint8_t *)&gLineCoding0, 7);
            }
            if (buf[4] == 2) {  /* 介面2: VCOM1的類別請求 */
                USBD_MemCopy((uint8_t *)(USBD_BUF_BASE +
USBD_GET_EP_BUF_ADDR(EP0)), (uint8_t *)&gLineCoding1, 7);
            }

            /* Data stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 7);
            /* Status stage */
```

```
            USBD_PrepareCtrlOut(0,0);
            break;
        }
        default: {
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
    }
    else {
    /* Host -> Device */
    switch (buf[1]) {
       case SET_CONTROL_LINE_STATE: {                        /* VCOM的類別請求 */
            if (buf[4] == 0) {  /* 介面0: VCOM0的類別請求 */
                gCtrlSignal0 = buf[3];
                gCtrlSignal0 = (gCtrlSignal0 << 8) | buf[2];
            }
            if (buf[4] == 2) {  /* 介面2: VCOM1的類別請求 */
                gCtrlSignal1 = buf[3];
                gCtrlSignal1 = (gCtrlSignal1 << 8) | buf[2];
            }

            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;
        }        case SET_LINE_CODE: {                        /* VCOM的類別請求 */
            if (buf[4] == 0)  /* 介面0: VCOM0的類別請求 */
                USBD_PrepareCtrlOut((uint8_t *)&gLineCoding0, 7);
            if (buf[4] == 2)  /* 介面2: VCOM1的類別請求 */
                USBD_PrepareCtrlOut((uint8_t *)&gLineCoding1, 7);

            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
         }
         default: {
            // Stall
            /* Setup error, stall the device */
```

```
            USBD_SetStall(0);
            break;
        }
      }
    }
}
void EP2_Handler(void)  /* BULK IN handler */
{
    gu32TxSize = 0;
}

void EP3_Handler(void)  /* BULK OUT handler */
{
    /* Bulk OUT */
    gu32RxSize = USBD_GET_PAYLOAD_LEN(EP3);
    gpu8RxBuf = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP3));

    /* Set a flag to indicate bulk out ready */
    gi8BulkOutReady = 1;
}

void EP6_Handler(void)  /* BULK OUT handler */
{
    /* Bulk OUT */
    gu32RxSize1 = USBD_GET_PAYLOAD_LEN(EP6);
    gpu8RxBuf1 = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP6));

    /* Set a flag to indicate bulk out ready */
    gi8BulkOutReady1 = 1;
}

void EP7_Handler(void) /* BULK IN handler */
{
    gu32TxSize1 = 0;
}
```

第一次執行此範例與電腦連接時需要安裝驅動程式，透過此範例搭配的INF檔案來安裝CDC驅動程式，成功安裝驅動程式後，硬體的管理員中就會出現兩個新的Nuvoton虛擬COM port，如圖 25分別為COM 26及COM 25。使用者可以利用網路

上免費的終端機連線工具分別來與兩個COM port做資料傳輸。執行此範例使用兩個終端機連線工具，選擇所對應的COM 26及COM 25，使用者可以在終端機連線工具上看到自己從鍵盤輸入的特定字元。圖 27為執行Dual VCOM範例的執行結果。

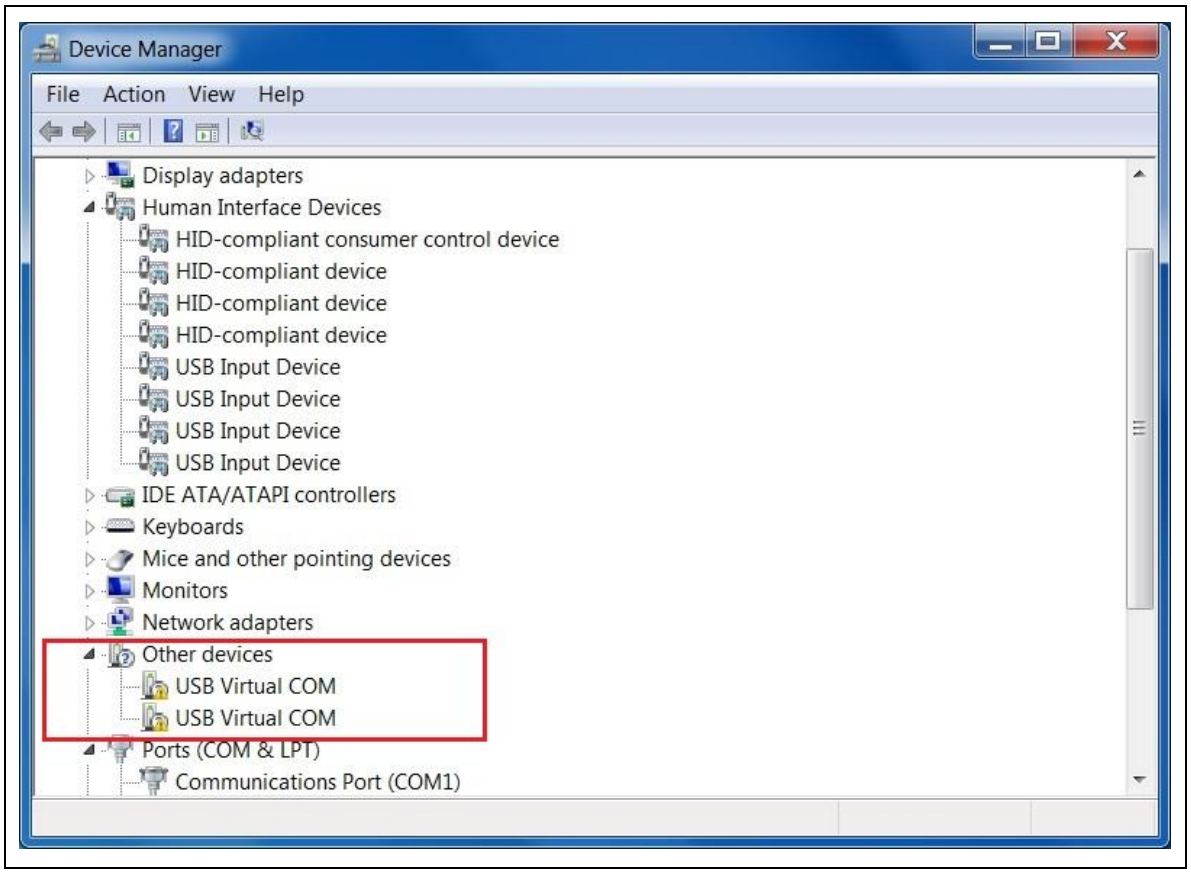
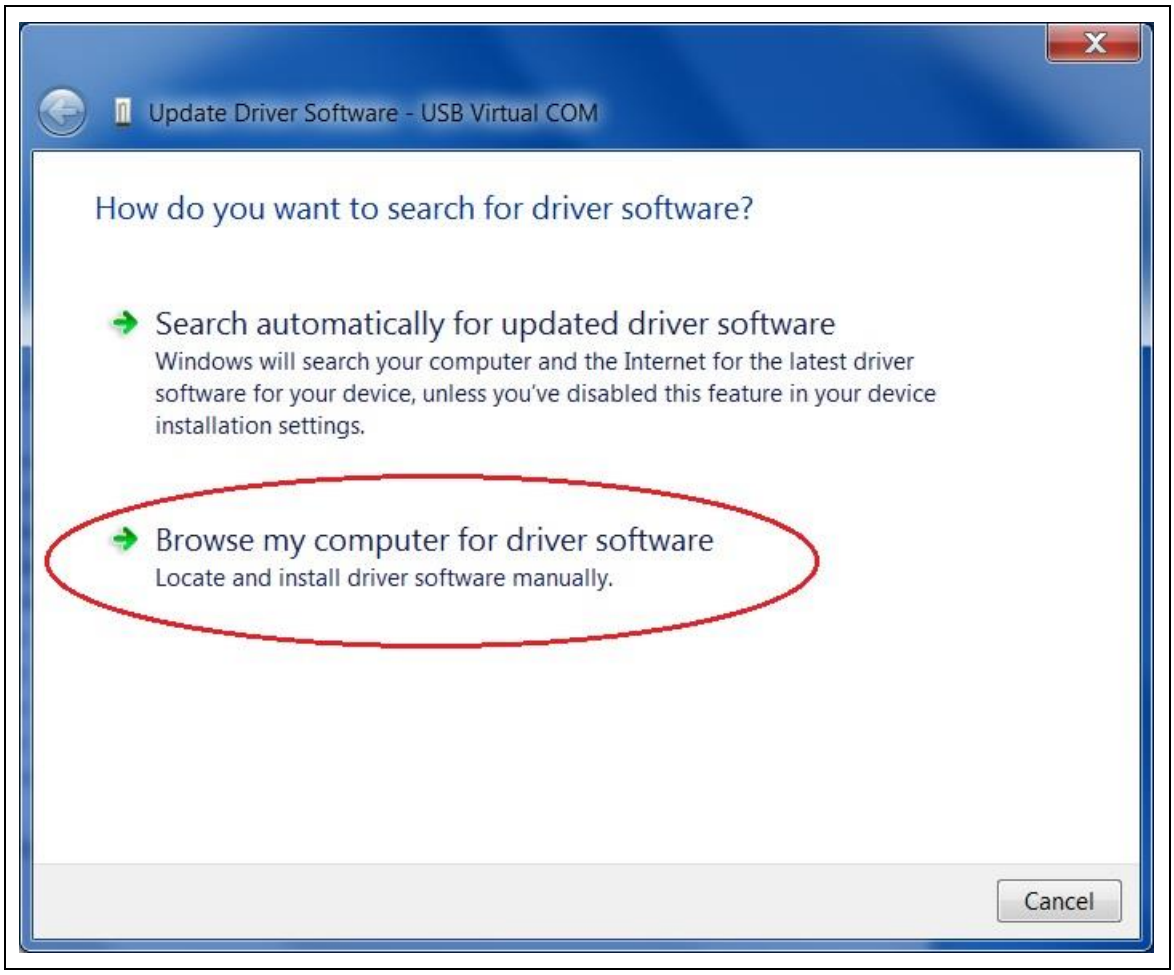


圖 23 未安裝驅動程式(Dual VCOM 複合裝置)


Update Driver Software - USB Virtual COM
How do you want to search for driver software?
Search automatically for updated driver software
Windows will search your computer and the Internet for the latest driver software for your device, unless you've disabled this feature in your device installation settings.
Browse my computer for driver software
Locate and install driver software manually.
Cancel

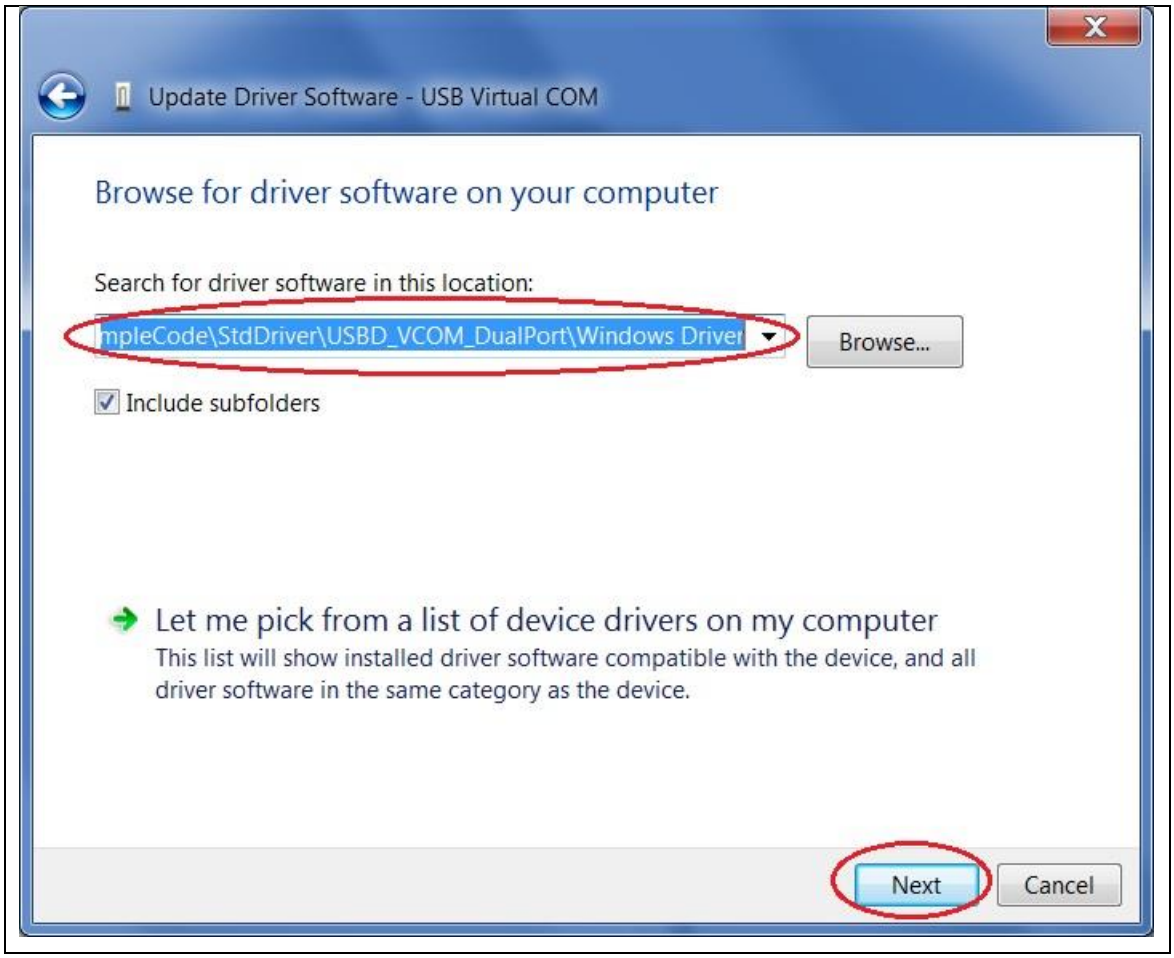

Update Driver Software - USB Virtual COM
Browse for driver software on your computer
Search for driver software in this location:
mpleCode\StdDriver\USBD_VCOM_DualPort\Windows Driver
Browse...
Include subfolders
Let me pick from a list of device drivers on my computer
This list will show installed driver software compatible with the device, and all driver software in the same category as the device.
Next
Cancel

圖 24 安裝 INF

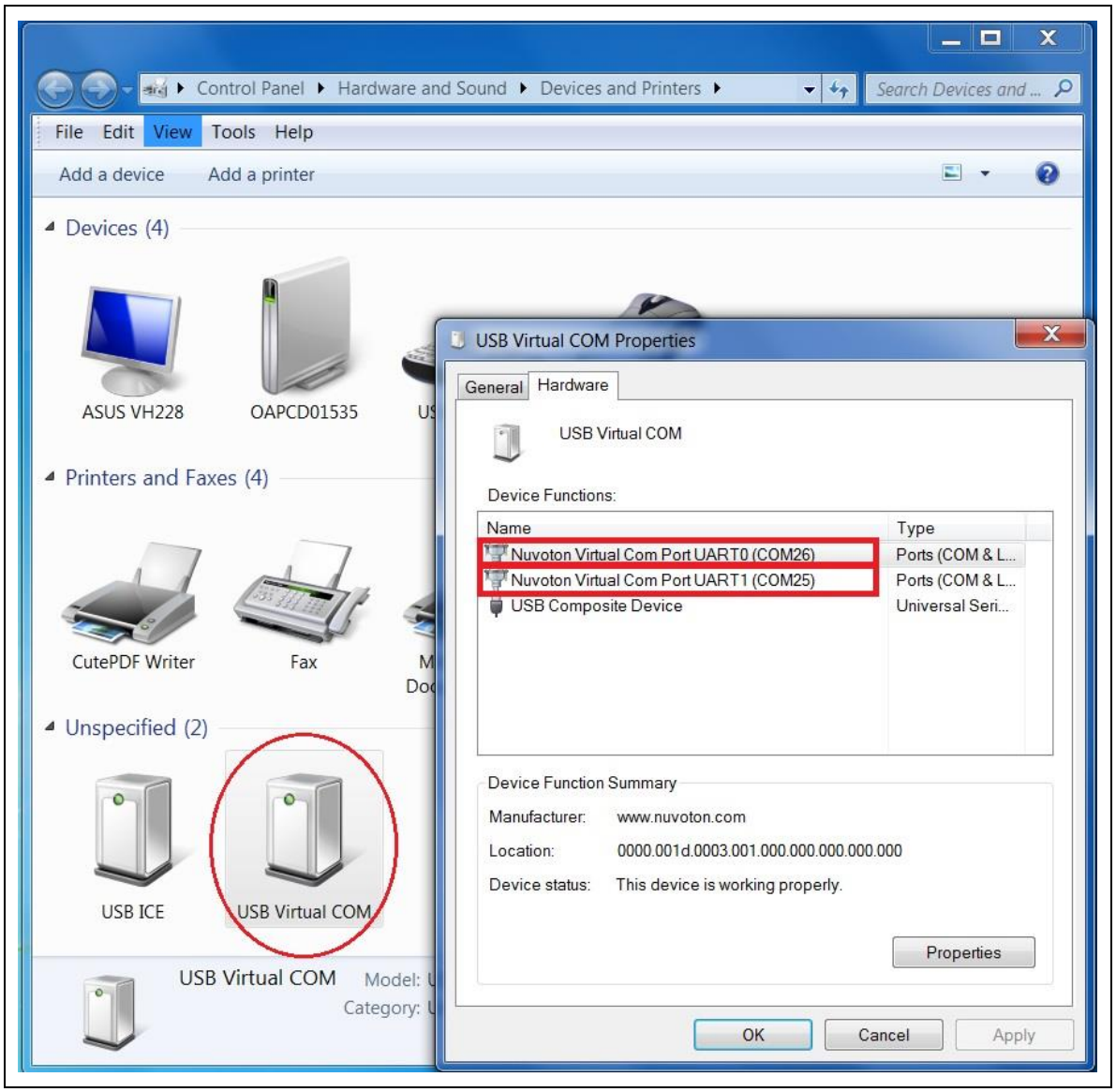


圖 25 Dual VCOM 複合裝置

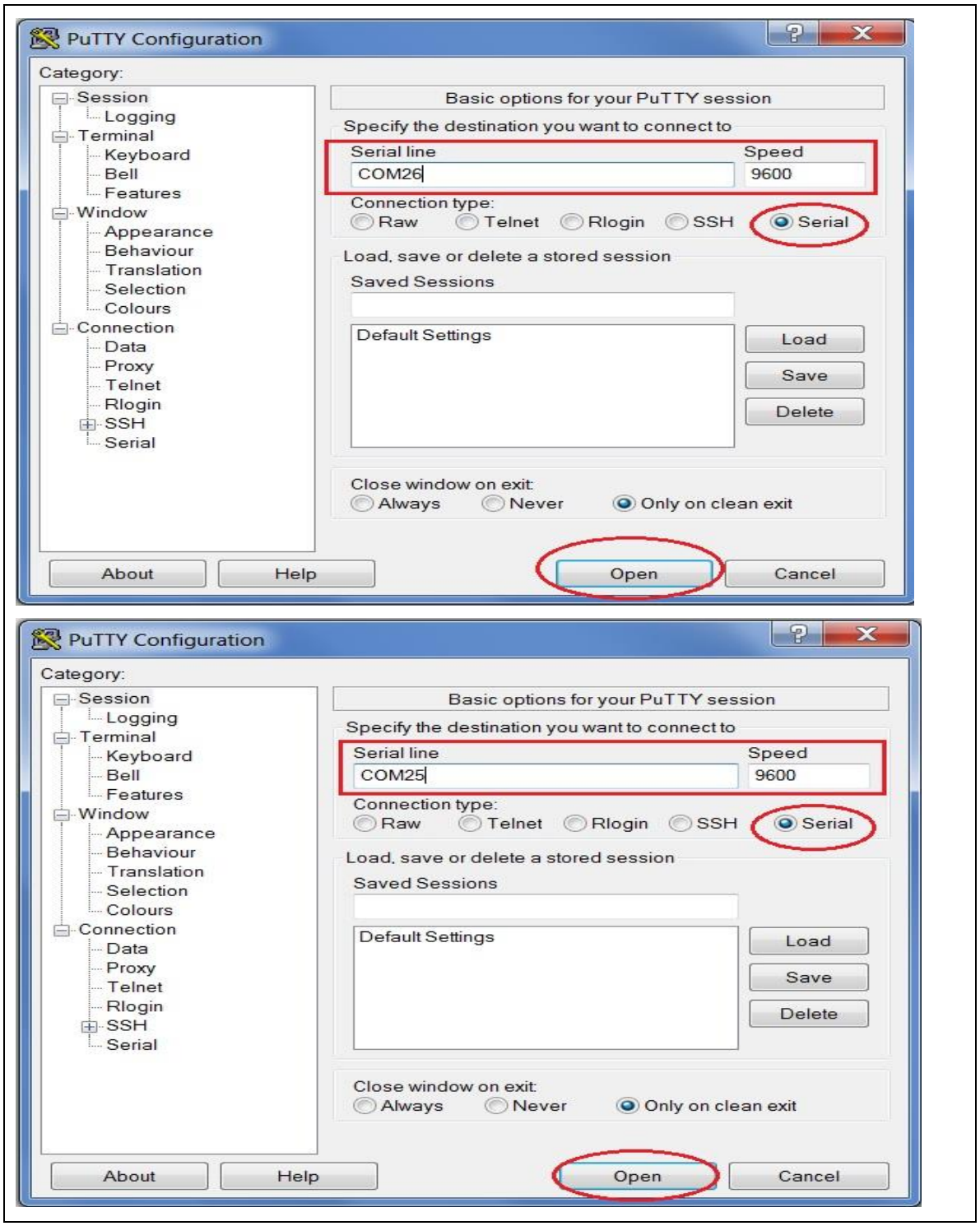


圖 26 Dual VCOM 的 COM0 及 COM1 設定

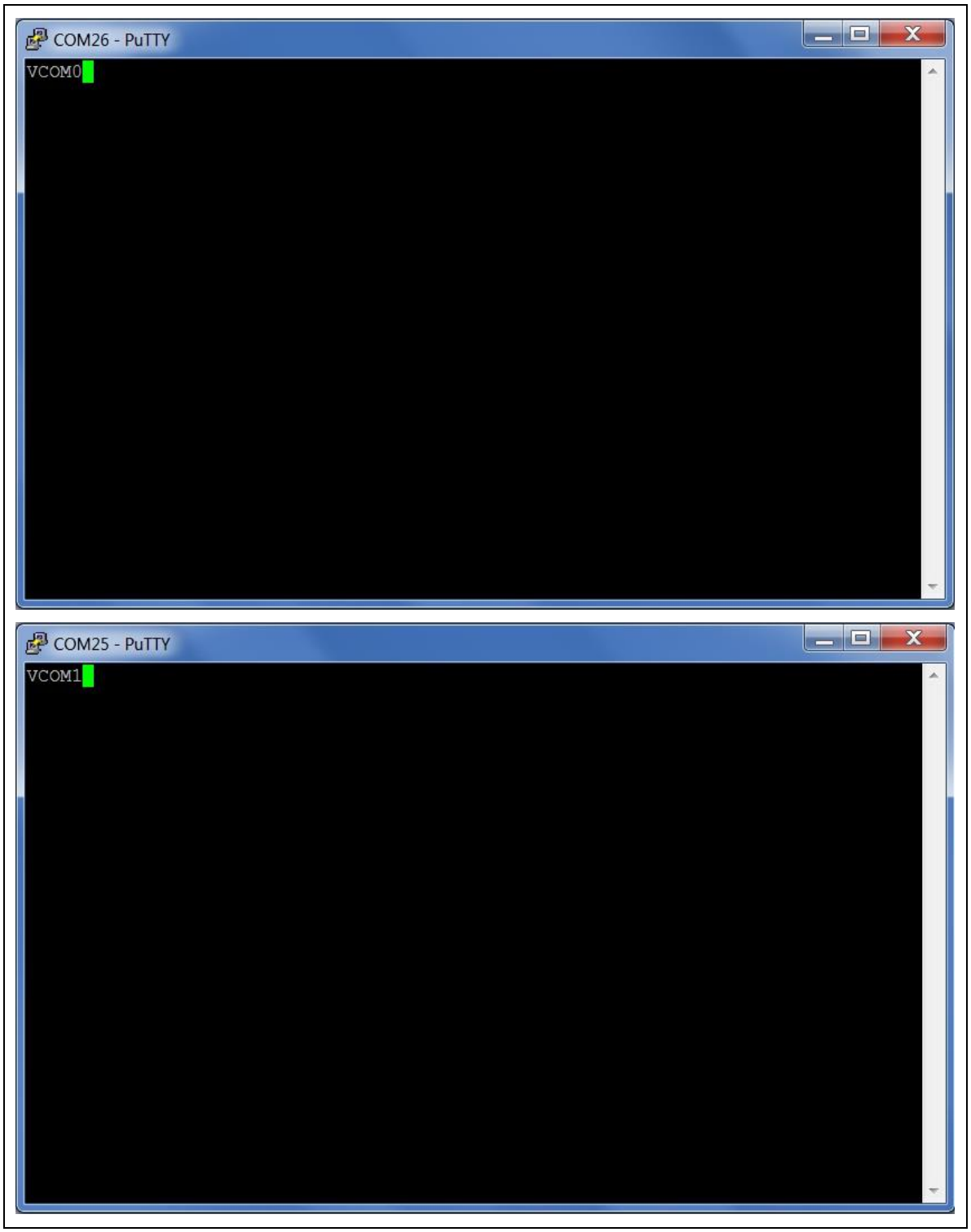


圖 27 Dual VCOM 範例

3.3.8 Audio + HID Transfer

這個複合裝置範例是結合Audio和HID Transfer，圖 28說明Audio和HID Transfer複合裝置範例的架構。首先可以分別從BSP\SampleCode\StdDriver\Nu-LB-NANO130分別取得Audio和從BSP\SampleCode\StdDriver取得HID Transfer這兩個獨立功能的範例程式，使用者可以藉由使用Audio的範例程式並結合HID Transfer的範例程式來實作出新的Audio和HID Transfer的複合裝置範例。

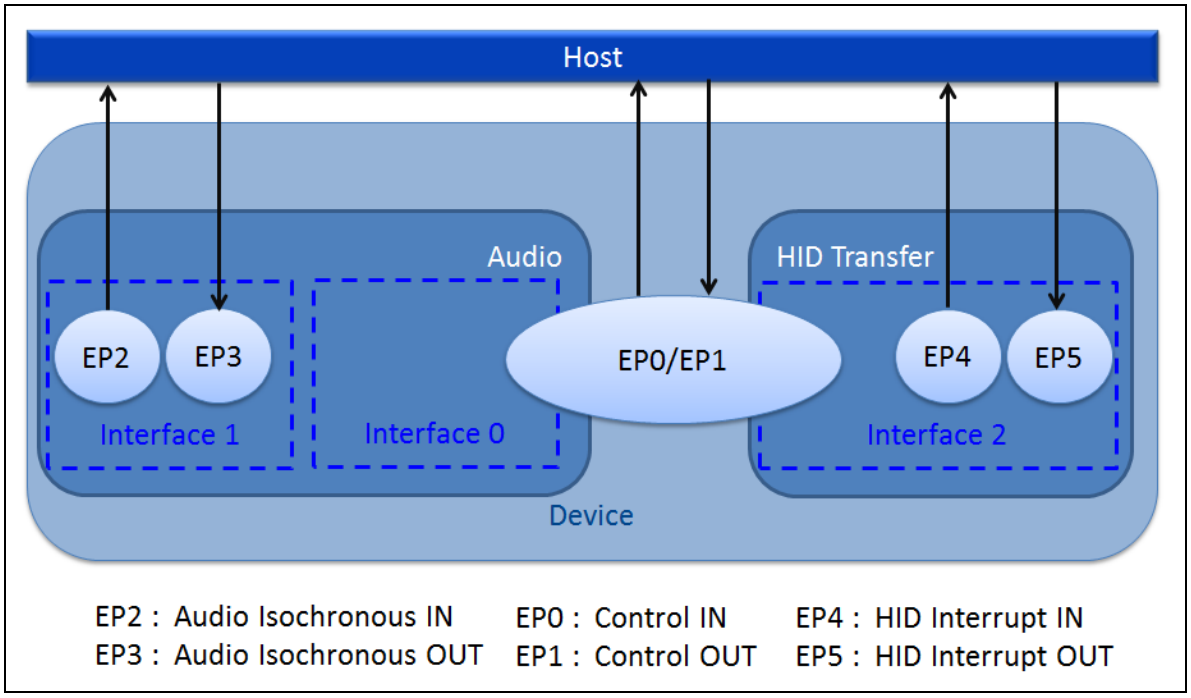


圖 28 Audio + HID Transfer複合裝置的架構

開發步驟如下：

步驟1：根據裝置的類別來配置端點和其對應的IN/OUT Buffer，此範例配置六個端點EP0~EP5。EP0_BUF_BASE~EP6_BUF_BASE分別為各個端點可以使用的Buffer起始位址。EP0被配置為Control IN，EP1被配置為Control OUT，EP2被配置為Isochronous IN，EP3被配置為Isochronous OUT，EP4被配置為Interrupt IN，EP5被配置為Interrupt OUT，除了EP2最大封包長度為32Bytes，其他端點的最大封包長度皆為64Bytes。下面的程式碼顯示Endpoint的設定。

```
/* 端點最大封包長度 */
#define EP0_MAX_PKT_SIZE    64
#define EP1_MAX_PKT_SIZE    EP0_MAX_PKT_SIZE
#define EP2_MAX_PKT_SIZE    REC_RATE*REC_CHANNELS*2/1000
#define EP3_MAX_PKT_SIZE    PLAY_RATE*PLAY_CHANNELS*2/1000
#define EP4_MAX_PKT_SIZE    64
#define EP5_MAX_PKT_SIZE    64
```

```
/* 各個端點在USB Buffer的偏移值 */
#define SETUP_BUF_BASE  0
#define SETUP_BUF_LEN   8
#define EP0_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP0_BUF_LEN      EP0_MAX_PKT_SIZE
#define EP1_BUF_BASE    (SETUP_BUF_BASE + SETUP_BUF_LEN)
#define EP1_BUF_LEN      EP1_MAX_PKT_SIZE
#define EP2_BUF_BASE    (EP1_BUF_BASE + EP1_BUF_LEN)
#define EP2_BUF_LEN      EP2_MAX_PKT_SIZE
#define EP3_BUF_BASE    (EP2_BUF_BASE + EP2_BUF_LEN)
#define EP3_BUF_LEN      EP3_MAX_PKT_SIZE
#define EP4_BUF_BASE    (EP3_BUF_BASE + EP3_BUF_LEN)
#define EP4_BUF_LEN      EP4_MAX_PKT_SIZE
#define EP5_BUF_BASE    (EP4_BUF_BASE + EP4_BUF_LEN)
#define EP5_BUF_LEN      EP5_MAX_PKT_SIZE

void UAC_Init(void)
{
    /* Init setup packet buffer */
    /* Buffer for setup packet -> [0 ~ 0x7] */
    USBD->BUFSEG = SETUP_BUF_BASE;

    /*****************************************************/
    /* EP0 ==> control IN endpoint, address 0 */
    USBD_CONFIG_EP(EP0, USBD_CFG_CSTALL | USBD_CFG_EPMODE_IN);
    /* Buffer range for EP0 */
    USBD_SET_EP_BUF_ADDR(EP0, EP0_BUF_BASE);

    /* EP1 ==> control OUT endpoint, address 0 */
    USBD_CONFIG_EP(EP1, USBD_CFG_CSTALL | USBD_CFG_EPMODE_OUT);
    /* Buffer range for EP1 */
    USBD_SET_EP_BUF_ADDR(EP1, EP1_BUF_BASE);

    /*****************************************************/
    /* EP2 ==> Isochronous IN endpoint, address 1 */
    USBD_CONFIG_EP(EP2, USBD_CFG_EPMODE_IN | ISO_IN_EP_NUM |
USBD_CFG_TYPE_ISO);
    /* Buffer offset for EP2 */
```

```
    USBD_SET_EP_BUF_ADDR(EP2, EP2_BUF_BASE);

    /*****************************************************/
    /* EP3 ==> Isochronous OUT endpoint, address 2 */
    USBD_CONFIG_EP(EP3, USBD_CFG_EPMODE_OUT | ISO_OUT_EP_NUM |
USBD_CFG_TYPE_ISO);
    /* Buffer offset for EP3 */
    USBD_SET_EP_BUF_ADDR(EP3, EP3_BUF_BASE);
    /* trigger receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);

    /*****************************************************/
    /* EP4 ==> Interrupt IN endpoint, address 3 */
    USBD_CONFIG_EP(EP4, USBD_CFG_EPMODE_IN | INT_IN_EP_NUM);
    /* Buffer range for EP4 */
    USBD_SET_EP_BUF_ADDR(EP4, EP4_BUF_BASE);

    /* EP5 ==> Interrupt OUT endpoint, address 4 */
    USBD_CONFIG_EP(EP5, USBD_CFG_EPMODE_OUT | INT_OUT_EP_NUM);
    /* Buffer range for EP5 */
    USBD_SET_EP_BUF_ADDR(EP5, EP5_BUF_BASE);
    /* trigger to receive OUT data */
    USBD_SET_PAYLOAD_LEN(EP5, EP5_MAX_PKT_SIZE);
}
```

步驟2：設備、配置、介面、端點和字串描述元都包含在"descriptor.c"中，使用者在此檔案中加入Audio和HID Transfer這兩個類別的各種描述元。因為此範例屬於多重介面的複合裝置，需要額外使用介面關聯描述元(IAD)來定義多重介面功能，因此設備描述元中的三個欄位(bDeviceClass、bDeviceSubClass及bDeviceProtocol)的值分別為：0xEF, 0x02, 0x01。

```
/* USB設備描述元， VID和PID決定USB主機要安裝哪個驅動程式 */
uint8_t gu8DeviceDescriptor[] = {
    LEN_DEVICE,     /* bLength */
    DESC_DEVICE,    /* bDescriptorType */
    0x10, 0x01,     /* bcdUSB */
    0xEF,               /* bDeviceClass: IAD*/
    0x02,               /* bDeviceSubClass */
    0x01,               /* bDeviceProtocol */
    EP0_MAX_PKT_SIZE,   /* bMaxPacketSize0 */
```

```
    /* idVendor */
    USBD_VID & 0x00FF,
    (USBD_VID & 0xFF00) >> 8,
    /* idProduct */
    USBD_PID & 0x00FF,
    (USBD_PID & 0xFF00) >> 8,
    0x00, 0x01,     /* bcdDevice */
    0x01,           /* iManufacture */
    0x02,           /* iProduct */
    0x00,           /* iSerialNumber - no serial */
    0x01            /* bNumConfigurations */
}

/* USB 組態描述元，它包含介面、HID及端點描述元 */
uint8_t gu8ConfigDescriptor[] = {
/* 組態描述元 */
    LEN_CONFIG,     /* bLength */
    DESC_CONFIG,    /* bDescriptorType */
    0xE8, 0x00,     /* wTotalLength */
    0x04,           /* bNumInterfaces */
    0x01,           /* bConfigurationValue */
    0x00,           /* iConfiguration */
    0x80,           /* bmAttributes */
    0x20,           /* Max power */

/* 介面關聯描述元(IAD) */
    0x08,    // bLength: Interface Descriptor size
    0x0B,    // bDescriptorType: IAD
    0x00,    // bFirstInterface
    0x03,    // bInterfaceCount
    0x01,    // bFunctionClass(0x01): AUDIO */
    0x01,    // bFunctionSubClass
    0x20,    // bFunctionProtocol
    0x00,    // iFunction

    /* Standard AC inteface */
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
    0x00,           /* bInterfaceNumber */
```

```
    0x00,           /* bAlternateSetting */
    0x00,           /* bNumEndpoints */
    0x01,           /* bInterfaceClass:AUDIO */
    0x01,           /* bInterfaceSubClass:AUDIOCONTROL */
    0x00,           /* bInterfaceProtocol */
    0x00,           /* iInterface */

    /* Class-spec AC interface descriptor */
    0x0A,           /* bLength */
    0x24,           /* bDescriptorType:CS_INTERFACE */
    0x01,           /* bDescriptorSubType:HEADER */
    0x00, 0x01,     /* bcdADC:1.0 */
    0x46, 0x00,     /* wTotalLength */
    0x02,           /* bInCollection */
    0x01,           /* baInterfaceNr(1) */
    0x02,           /* baInterfaceNr(n) */

    /* TID 1: Input for usb streaming */
    0x0C,               /* bLength */
    0x24,               /* bDescriptorType:CS_INTERFACE */
    0x02,               /* bDescriptorSubType:INPUT_TERMINAL */
    0x01,               /* bTerminalID */
    0x01, 0x01,         /* wTerminalType: 0x0101 usb streaming */
    0x00,               /* bAssocTerminal */
    PLAY_CHANNELS,      /* bNrChannels */
    PLAY_CH_CFG, 0x00,  /* wChannelConfig */
    0x00,               /* iChannelNames */
    0x00,               /* iTerminal */

    /* UNIT ID 5: Feature Unit */
    0x08,               /* bLength */
    0x24,               /* bDescriptorType */
    0x06,               /* bDescriptorSubType */
    REC_FEATURE_UNITID, /* bUnitID */
    0x04,               /* bSourceID */
    0x01,               /* bControlSize */
    0x03,               /* bmaControls(0) */
    0x00,               /* iFeature */
```

```
    /* TID 2: Output Terminal for usb streaming */
    0x09,               /* bLength */
    0x24,               /* bDescriptorType:CS_INTERFACE */
    0x03,               /* bDescriptorSubType:OUTPUT_TERMINAL */
    0x02,               /* bTerminalID */
    0x01, 0x01,         /* wTerminalType: 0x0101 usb streaming */
    0x00,               /* bAssocTerminal */
    REC_FEATURE_UNITID, /* bSourceID */
    0x00,               /* iTerminal */

    /* UNIT ID 6: Feature Unit */
    0x0A,               /* bLength */
    0x24,               /* bDescriptorType */
    0x06,               /* bDescriptorSubType */
    PLAY_FEATURE_UNITID, /* bUnitID */
    0x01,               /* bSourceID */
    0x01,               /* bControlSize */
    0x01,               /* bmaControls(0) */
    0x02,               /* bmaControls(0) */
    0x02,               /* bmaControls(0) */
    0x00,               /* iFeature */

    /* TID 3: Output for speaker */
    0x09,           /* bLength*/
    0x24,           /* bDescriptorType:CS_INTERFACE*/
    0x03,           /* bDescriptorSubType:OUTPUT_TERMINAL*/
    0x03,           /* bTerminalID*/
    0x01, 0x03,     /* wTerminalType: 0x0301 speaker*/
    0x00,           /* bAssocTerminal*/
    0x06,           /* bSourceID*/
    0x00,           /* iTerminal*/

    /* TID 4: Input Terminal for microphone */
    0x0C,               /* bLength */
    0x24,               /* bDescriptorType:CS_INTERFACE */
    0x02,               /* bDescriptorSubType:INPUT_TERMINAL*/
    0x04,               /* bTerminalID*/
    0x01, 0x02,         /* wTerminalType: 0x0201 microphone*/
    0x00,               /* bAssocTerminal*/
```

```
    REC_CHANNELS,       /* bNrChannels*/
    REC_CH_CFG, 0x00,   /* wChannelConfig*/
    0x00,               /* iChannelNames*/
    0x00,               /* iTerminal*/

    /* Standard AS interface 1, alternate 0 */
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
    0x01,           /* bInterfaceNumber */
    0x00,           /* bAlternateSetting */
    0x00,           /* bNumEndpoints */
    0x01,           /* bInterfaceClass:AUDIO */
    0x02,           /* bInterfaceSubClass:AUDIOSTREAMING */
    0x00,           /* bInterfaceProtocol */
    0x00,           /* iInterface */

    /* Standard AS interface 1, alternate 1 */
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
    0x01,           /* bInterfaceNumber */
    0x01,           /* bAlternateSetting */
    0x01,           /* bNumEndpoints */
    0x01,           /* bInterfaceClass:AUDIO */
    0x02,           /* bInterfaceSubClass:AUDIOSTREAMING */
    0x00,           /* bInterfaceProtocol */
    0x00,           /* iInterface */

   /* Class-spec AS interface, this interface's endpoint connect to TID 0x02
*/
    0x07,           /* bLength */
    0x24,           /* bDescriptorType:CS_INTERFACE */
    0x01,           /* bDescriptorSubType:AS_GENERAL */
    0x02,           /* bTernimalLink */
    0x01,           /* bDelay */
    0x01, 0x00,     /* wFormatTag:0x0001 PCM */

    /* Type I format type Descriptor */
    0x0B,           /* bLength */
    0x24,           /* bDescriptorType:CS_INTERFACE */
```

```
    0x02,           /* bDescriptorSubType:FORMAT_TYPE */
    0x01,           /* bFormatType:FORMAT_TYPE_I */
    REC_CHANNELS,   /* bNrChannels */
    0x02,           /* bSubFrameSize */
    0x10,           /* bBitResolution */
    0x01,           /* bSamFreqType : 0 continuous; 1 discrete */
    REC_RATE_LO,
    REC_RATE_MD,
    REC_RATE_HI,    /* Sample Frequency */

/* 端點2描述元: Isochronous IN，地址為ISO_IN_EP_NUM */
    0x09,                       /* bLength */
    0x05,                       /* bDescriptorType */
    ISO_IN_EP_NUM | EP_INPUT,   /* bEndpointAddress */
    0x0d,                       /* bmAttributes */
    EP2_MAX_PKT_SIZE,0x00,       /* wMaxPacketSize*/
    0x01,                       /* bInterval*/
    0x00,                       /* bRefresh*/
    0x00,                       /* bSynchAddress*/

    /* Class-spec AS ISO Audio Data endpoint Descriptor */
    0x07,           /* bLength */
    0x25,           /* bDescriptorType:CS_ENDPOINT */
    0x01,           /* bDescriptorSubType:EP_GENERAL */
    0x00,           /* bmAttributes */
    0x00,           /* bLockDelayUnits */
    0x00, 0x00,     /* wLockDelay */

    /* Standard AS interface 2, alternate 0 */
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
    0x02,           /* bInterfaceNumber */
    0x00,           /* bAlternateSetting */
    0x00,           /* bNumEndpoints */
    0x01,           /* bInterfaceClass:AUDIO */
    0x02,           /* bInterfaceSubClass:AUDIOSTREAMING */
    0x00,           /* bInterfaceProtocol */
    0x00,           /* iInterface */
```

```
    /* Standard AS interface 2, alternate 1 */
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
    0x02,           /* bInterfaceNumber */
    0x01,           /* bAlternateSetting */
    0x01,           /* bNumEndpoints */
    0x01,           /* bInterfaceClass:AUDIO */
    0x02,           /* bInterfaceSubClass:AUDIOSTREAMING */
    0x00,           /* bInterfaceProtocol */
    0x00,           /* iInterface */

    /* Class-spec AS inf this interface's endpoint connect to TID 0x01 */
    0x07,           /* bLength */
    0x24,           /* bDescriptorType:CS_INTERFACE */
    0x01,           /* bDescriptorSubType:AS_GENERAL */
    0x01,           /* bTernimalLink */
    0x01,           /* bDelay */
    0x01, 0x00,     /* wFormatTag:0x0001 PCM */

    /* Type I format type Descriptor */
    0x0B,           /* bLength */
    0x24,           /* bDescriptorType:CS_INTERFACE */
    0x02,           /* bDescriptorSubType:FORMAT_TYPE */
    0x01,           /* bFormatType:FORMAT_TYPE_I */
    PLAY_CHANNELS,  /* bNrChannels */
    0x02,           /* bSubFrameSize */
    0x10,           /* bBitResolution */
    0x01,           /* bSamFreqType : 0 continuous; 1 discrete */
    PLAY_RATE_LO,
    PLAY_RATE_MD,
    PLAY_RATE_HI,   /* Sample Frequency */

/* 端點3描述元: Isochronous OUT，地址為ISO_OUT_EP_NUM */
    0x09,                       /* bLength */
    0x05,                       /* bDescriptorType */
    ISO_OUT_EP_NUM | EP_OUTPUT, /* bEndpointAddress */
    0x0d,                       /* bmAttributes */
    EP3_MAX_PKT_SIZE, 0x00,     /* wMaxPacketSize */
    0x01,                       /* bInterval */
```

```
    0x00,                       /* bRefresh */
    0x00,                       /* bSynchAddress */

    /* Class-spec AS ISO Audio Data endpoint Descriptor */
    0x07,           /* bLength */
    0x25,           /* bDescriptorType:CS_ENDPOINT */
    0x01,           /* bDescriptorSubType:EP_GENERAL */
    0x80,           /* bmAttributes */
    0x00,           /* bLockDelayUnits */
    0x00, 0x00,     /* wLockDelay */

/* 介面3 : HID Transfer介面描述元*/
    LEN_INTERFACE,  /* bLength */
    DESC_INTERFACE, /* bDescriptorType */
    0x03,           /* bInterfaceNumber */
    0x00,           /* bAlternateSetting */
    0x02,           /* bNumEndpoints */
    0x03,           /* bInterfaceClass */
    0x00,           /* bInterfaceSubClass */
    0x00,           /* bInterfaceProtocol */
    0x00,           /* iInterface */

/* HID描述元 */
    LEN_HID,        /* Size of this descriptor in UINT8s. */
    DESC_HID,       /* HID descriptor type. */
    0x10, 0x01,     /* HID Class Spec. release number. */
    0x00,           /* H/W target country. */
    0x01,           /* Number of HID class descriptors to follow. */
    DESC_HID_RPT,   /* Descriptor type. */
    /* Total length of report descriptor. */
    sizeof(HID_DeviceReportDescriptor) & 0x00FF,
    (sizeof(HID_DeviceReportDescriptor) & 0xFF00) >> 8,

/* 端點4描述元: 中斷IN，地址為INT_IN_EP_NUM */
    LEN_ENDPOINT,   /* bLength */
    DESC_ENDPOINT,  /* bDescriptorType */
    (INT_IN_EP_NUM | EP_INPUT), /* bEndpointAddress */
    EP_INT,         /* bmAttributes */
    /* wMaxPacketSize */
```

```
    EP4_MAX_PKT_SIZE & 0x00FF,
    (EP4_MAX_PKT_SIZE & 0xFF00) >> 8,
    1,              /* bInterval */

/* 端點5描述元: 中斷OUT，地址為INT_OUT_EP_NUM */
    LEN_ENDPOINT,   /* bLength */
    DESC_ENDPOINT,  /* bDescriptorType */
    (INT_OUT_EP_NUM | EP_OUTPUT),   /* bEndpointAddress */
    EP_INT,         /* bmAttributes */
    /* wMaxPacketSize */
    EP5_MAX_PKT_SIZE & 0x00FF,
    (EP5_MAX_PKT_SIZE & 0xFF00) >> 8,
    1,             /* bInterval */
};

/* HID Transfer Report描述元 */
uint8_t HID_DeviceReportDescriptor[] =
{
    0x06, 0x00, 0xFF,   // Usage Page = 0xFF00 (Vendor Defined Page 1)
    0x09, 0x01,         // Usage (Vendor Usage 1)
    0xA1, 0x01,         // Collection (Application)
    0x19, 0x01,         // Usage Minimum
    0x29, 0x40,         // Usage Maximum
    0x15, 0x00,         // Logical Minimum
    0x26, 0xFF, 0x00,   // Logical Maximum
    0x75, 0x08,         // Report Size: 8-bit field size
    0x95, 0x40,         // Report Count
    // an "Input", "Output", or "Feature" item)
    0x81, 0x00,         // Input (Data, Array, Abs)
   // above report size, count, logical min/max, and usage.
    0x19, 0x01,         // Usage Minimum
    0x29, 0x40,         // Usage Maximum
    0x91, 0x00,         // Output (Data, Array, Abs)
    0xC0                // End Collection
};

/* 字串描述元 */
uint8_t gu8StringLang[4] = {
    4,                                          /* bLength */
```

```
    DESC_STRING,                                /* bDescriptorType */
    0x09, 0x04
};

/* 字串描述元 */
uint8_t gu8VendorStringDesc[] = {
    16,
    DESC_STRING,
    'N', 0, 'u', 0, 'v', 0, 'o', 0, 't', 0, 'o', 0, 'n', 0
};

/* 字串描述元 */
uint8_t gu8ProductStringDesc[] = {
    36,
    DESC_STRING,
    'U', 0, 'S', 0, 'B', 0, ' ', 0, 'A', 0, 'u', 0, 'd', 0, 'i', 0, 'o', 0,
    ' ', 0, 'a', 0, 'n', 0, 'd', 0, ' ', 0,
    'H', 0, 'I', 0, 'D', 0
};

/* 因為此範例的HID為介面3，因此HID相關結構的Index皆為3 */
uint8_t *gu8UsbHidReport[4] = {
    NULL,
    NULL,
    NULL,
    HID_DeviceReportDescriptor,            /* HID transfer報告描述元 */
};

uint32_t gu32UsbHidReportLen[4] = {
    0,
    0,
    0,
    sizeof(HID_DeviceReportDescriptor),    /* HID transfer報告描述元的長度 */
};

uint32_t gu32ConfigHidDescIdx[4] = {
    0,
    0,
    0,
```

```
/* HID transfer描述元在組態描述元內的位置 */
    (sizeof(gu8ConfigDescriptor) - LEN_HID - (2 * LEN_ENDPOINT)),
};
```

步驟3：usbd_audio.c執行Audio/HID類別請求和端點資料處理，在Audio設備類別中定義必要的特殊類別請求有三個，HID設備定義六個類別請求，但是並不是所有的命令都必須回應。usbd_audio.c執行Audio/HID類別請求及端點EP2~EP5的Callback如下所示：

```
void UAC_ClassRequest(void)
{
    uint8_t buf[8];
    USBD_GetSetupPacket(buf);

    if (buf[0] & 0x80) {                /* 該命令數據傳輸方向是Device -> Host */
        switch (buf[1]) {
        case UAC_GET_CUR: {  /* 介面0: Audio的類別請求 */
            switch (buf[3]) {
            case MUTE_CONTROL: {  /* 介面0: Audio的類別請求 */
                if (REC_FEATURE_UNITID == buf[5])
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) =
g_usbd_RecMute;
                else if (PLAY_FEATURE_UNITID == buf[5])
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) =
g_usbd_PlayMute;

                /* Data stage */
                USBD_SET_DATA1(EP0);
                USBD_SET_PAYLOAD_LEN(EP0, 1);
                break;
            }
            case VOLUME_CONTROL: {
                if (REC_FEATURE_UNITID == buf[5]) {
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) =
g_usbd_RecVolume;
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
g_usbd_RecVolume >> 8;
                } else if (PLAY_FEATURE_UNITID == buf[5]) {
                    if(buf[2] == 1) {
                        M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) =
g_usbd_PlayVolumeL;
```

```
                        M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
g_usbd_PlayVolumeL >> 8;
                    } else {
                        M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) =
g_usbd_PlayVolumeR;
                        M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
g_usbd_PlayVolumeR >> 8;
                    }
                }
                /* Data stage */
                USBD_SET_DATA1(EP0);
                USBD_SET_PAYLOAD_LEN(EP0, 2);
                break;
            }
            default: {
                /* Setup error, stall the device */
                USBD_SetStall(0);
            }
            }
            // Trigger next Control Out DATA1 Transaction.
            /* Status stage */
            USBD_PrepareCtrlOut(0,0);
            break;
        }

        case UAC_GET_MIN: {  /* 介面0: Audio的類別請求 */
            switch (buf[3]) {
            case VOLUME_CONTROL: {
                if (REC_FEATURE_UNITID == buf[5]) {
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) =
g_usbd_RecMinVolume;
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
g_usbd_RecMinVolume >> 8;
                } else if (PLAY_FEATURE_UNITID == buf[5]) {
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) =
g_usbd_PlayMinVolume;
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
g_usbd_PlayMinVolume >> 8;
                }
                /* Data stage */
                USBD_SET_DATA1(EP0);
```

```
                USBD_SET_PAYLOAD_LEN(EP0, 2);
                break;
            }
            default:
                /* STALL control pipe */
                USBD_SetStall(0);
            }
            // Trigger next Control Out DATA1 Transaction.
            /* Status stage */
            USBD_PrepareCtrlOut(0,0);
            break;
        }

        case UAC_GET_MAX: {  /* 介面0: Audio的類別請求 */
            switch (buf[3]) {
            case VOLUME_CONTROL: {
                if (REC_FEATURE_UNITID == buf[5]) {
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) =
g_usbd_RecMaxVolume;
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
g_usbd_RecMaxVolume >> 8;
                } else if (PLAY_FEATURE_UNITID == buf[5]) {
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) =
g_usbd_PlayMaxVolume;
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
g_usbd_PlayMaxVolume >> 8;
                }
                /* Data stage */
                USBD_SET_DATA1(EP0);
                USBD_SET_PAYLOAD_LEN(EP0, 2);
                break;
            }
            default:
                /* STALL control pipe */
                USBD_SetStall(0);
            }
            // Trigger next Control Out DATA1 Transaction.
            /* Status stage */
            USBD_PrepareCtrlOut(0,0);
            break;
```

```
        }

        case UAC_GET_RES: {  /* 介面0: Audio的類別請求 */
            switch (buf[3]) {
            case VOLUME_CONTROL: {
                if (REC_FEATURE_UNITID == buf[5]) {
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) =
g_usbd_RecResVolume;
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
g_usbd_RecResVolume >> 8;
                } else if (PLAY_FEATURE_UNITID == buf[5]) {
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0)) =
g_usbd_PlayResVolume;
                    M8(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP0) + 1) =
g_usbd_PlayResVolume >> 8;
                }
                /* Data stage */
                USBD_SET_DATA1(EP0);
                USBD_SET_PAYLOAD_LEN(EP0, 2);
                break;
            }
            default:
                /* STALL control pipe */
                USBD_SetStall(0);
            }
            // Trigger next Control Out DATA1 Transaction.
            /* Status stage */
            USBD_PrepareCtrlOut(0,0);
            break;
        }

        default: {
            /* Setup error, stall the device */
            USBD_SetStall(0);
        }
        }
    } else {
        // Host to device
        switch (buf[1]) {
        case UAC_SET_CUR: {  /* 介面0: Audio的類別請求 */
```

```
            switch (buf[3]) {
            case MUTE_CONTROL:
                if (REC_FEATURE_UNITID == buf[5])
                    USBD_PrepareCtrlOut((uint8_t *)&g_usbd_RecMute, buf[6]);
                else if (PLAY_FEATURE_UNITID == buf[5]) {
                    USBD_PrepareCtrlOut((uint8_t *)&g_usbd_PlayMute, buf[6]);
                }
                /* Status stage */
                USBD_SET_DATA1(EP0);
                USBD_SET_PAYLOAD_LEN(EP0, 0);
                break;

            case VOLUME_CONTROL:  /* 介面0: Audio的類別請求 */
                if (REC_FEATURE_UNITID == buf[5])
                    USBD_PrepareCtrlOut((uint8_t *)&g_usbd_RecVolume, buf[6]);
                else if (PLAY_FEATURE_UNITID == buf[5])
                {
                    if (buf[2] == 1)
                    {
                        USBD_PrepareCtrlOut((uint8_t *)&g_usbd_PlayVolumeL,
buf[6]);
                        if(g_usbd_PlayVolumeL & 0x8000)
                            g_usbd_PlayVolumeL = (g_usbd_PlayVolumeL & 0x7FFF)
>> 8;
                        else
                            g_usbd_PlayVolumeL = (g_usbd_PlayVolumeL >> 7);
                        bPlayVolumeLAdjust = TRUE; //ATOM_I2C_WriteWAU8822(11,
i16PlayVolumeL | 0x100);   /* Set left DAC volume */
                    } else {
                        USBD_PrepareCtrlOut((uint8_t *)&g_usbd_PlayVolumeR,
buf[6]);
                        if(g_usbd_PlayVolumeR & 0x8000)
                            g_usbd_PlayVolumeR = (g_usbd_PlayVolumeR & 0x7FFF)
>> 8;
                        else
                            g_usbd_PlayVolumeR = (g_usbd_PlayVolumeR >> 7);
                        bPlayVolumeRAdjust = TRUE; //ATOM_I2C_WriteWAU8822(12,
i16PlayVolumeR | 0x100);   /* Set right DAC volume */
                    }
                }
                /* Status stage */
```

```
                USBD_SET_DATA1(EP0);
                USBD_SET_PAYLOAD_LEN(EP0, 0);
                break;

            default:
                /* STALL control pipe */
                USBD_SetStall(0);
                break;
            }
            break;
        }
        /* 介面2: HID的類別請求 */
        case SET_REPORT: {
            if (buf[3] == 3) {
                /* Request Type = Feature */
                USBD_SET_DATA1(EP1);
                USBD_SET_PAYLOAD_LEN(EP1, 0);
            }
            break;
        }
        /* 介面2: HID的類別請求 */
        case SET_IDLE: {
            /* Status stage */
            USBD_SET_DATA1(EP0);
            USBD_SET_PAYLOAD_LEN(EP0, 0);
            break;
        }
        /* 介面2: HID的類別請求 */
        case SET_PROTOCOL:                                          /* 不處理 */
        default: {
            // Stall
            /* Setup error, stall the device */
            USBD_SetStall(0);
            break;
        }
        }
    }
}
```

```
/* Record */
void EP2_Handler(void)  /* Isochronous IN handler */
{
    /* ISO IN transfer ACK */
    if (g_usbd_UsbAudioState == UAC_START_AUDIO_RECORD) {
        UAC_DeviceEnable(UAC_MICROPHONE);
        g_usbd_UsbAudioState = UAC_PROCESSING_AUDIO_RECORD;
    } else if (g_usbd_UsbAudioState == UAC_PROCESSING_AUDIO_RECORD)
        g_usbd_UsbAudioState = UAC_BUSY_AUDIO_RECORD;

    if (g_usbd_UsbAudioState == UAC_BUSY_AUDIO_RECORD)
        UAC_SendRecData();
    else
        USBD_SET_PAYLOAD_LEN(EP2, 0);
}

/* Play */
void EP3_Handler(void)  /* Isochronous OUT handler */
{
    /* ISO OUT transfer ACK */
    UAC_GetPlayData((int16_t *)((uint32_t)USBD_BUF_BASE +
USBD_GET_EP_BUF_ADDR(EP3)), (int16_t)USBD_GET_PAYLOAD_LEN(EP3));

    USBD_SET_PAYLOAD_LEN(EP3, EP3_MAX_PKT_SIZE);
}

void EP4_Handler(void)  /* Interrupt IN handler */
{
    HID_SetInReport();
}

void EP5_Handler(void)  /* Interrupt OUT handler */
{
    uint8_t *ptr;
    /* Interrupt OUT */
    ptr = (uint8_t *)(USBD_BUF_BASE + USBD_GET_EP_BUF_ADDR(EP5));
    HID_GetOutReport(ptr, USBD_GET_PAYLOAD_LEN(EP5));
    USBD_SET_PAYLOAD_LEN(EP5, EP5_MAX_PKT_SIZE);
}
```

執行此範例當裝置列舉成功時，可在電腦的系統管理中看到Audio + HID Transfer的複合裝置，不需另外安裝驅動程式，如圖 29所示。電腦主機端會把此裝置看作一個USB Audio音效裝置及HID裝置。

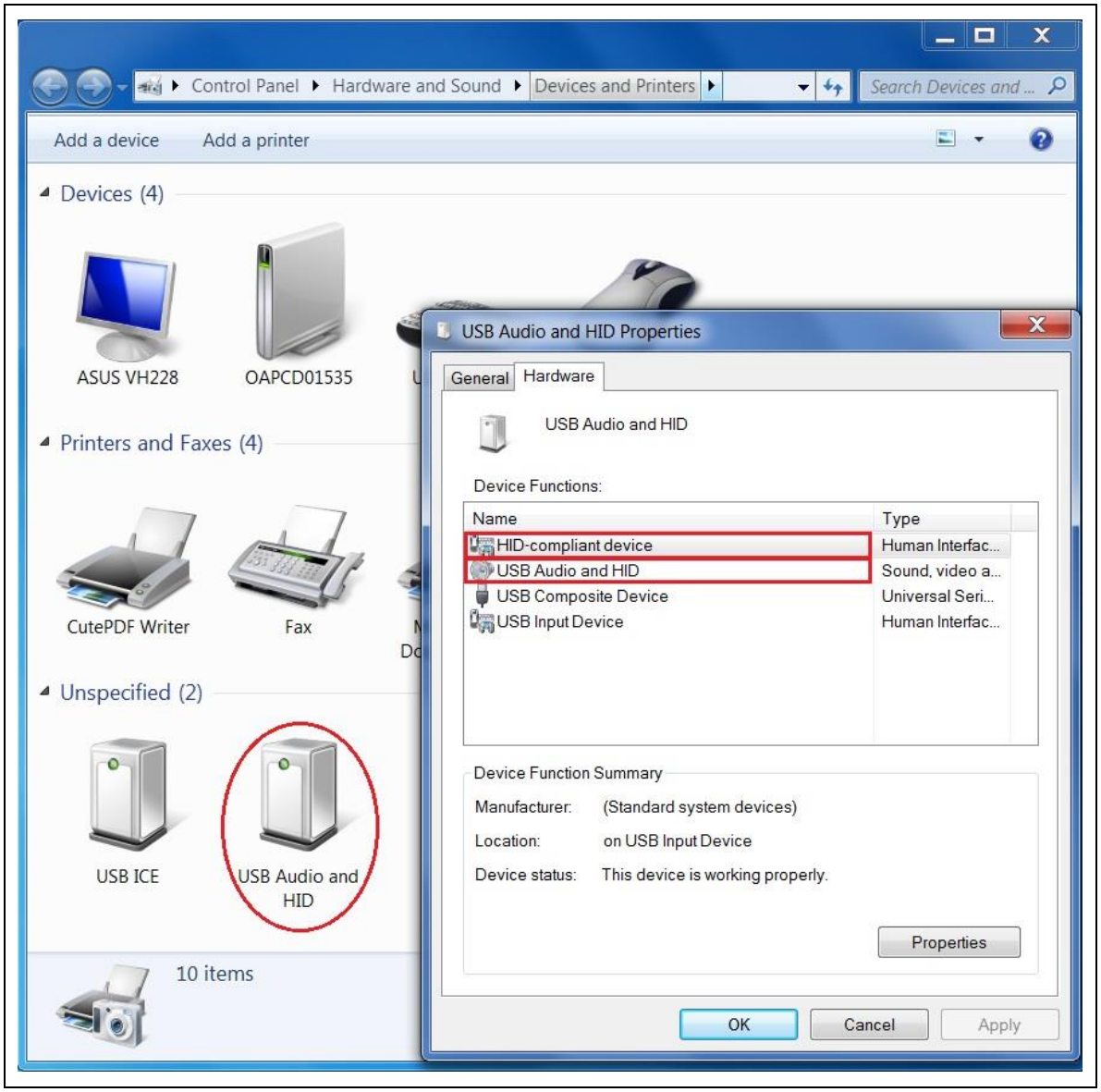


圖 29 Audio + HID Transfer 複合裝置

4 Windows Tool

4.1 概述

本章節介紹HID Transfer專用的主機端應用程式，利用Microsoft的Visual C++來開發。HID Transfer應用程式簡單示範如何透過Report來讀寫HID設備，使用者可以利用此應用程式來驗證本文中的HID Transfer裝置是否可以正常操作；也可以參考此範例的程式碼來開發其他的HID應用。

4.2 Demo

HID Transfer專用主機端應用程式的執行檔放在WindowsTool\debug\HIDTransferTest.exe，執行此程式可以驗證HID Transfer裝置是否列舉成功並且可以正常操作使用。執行應用程式需要先輸入裝置的PID，執行結果如下圖所示。

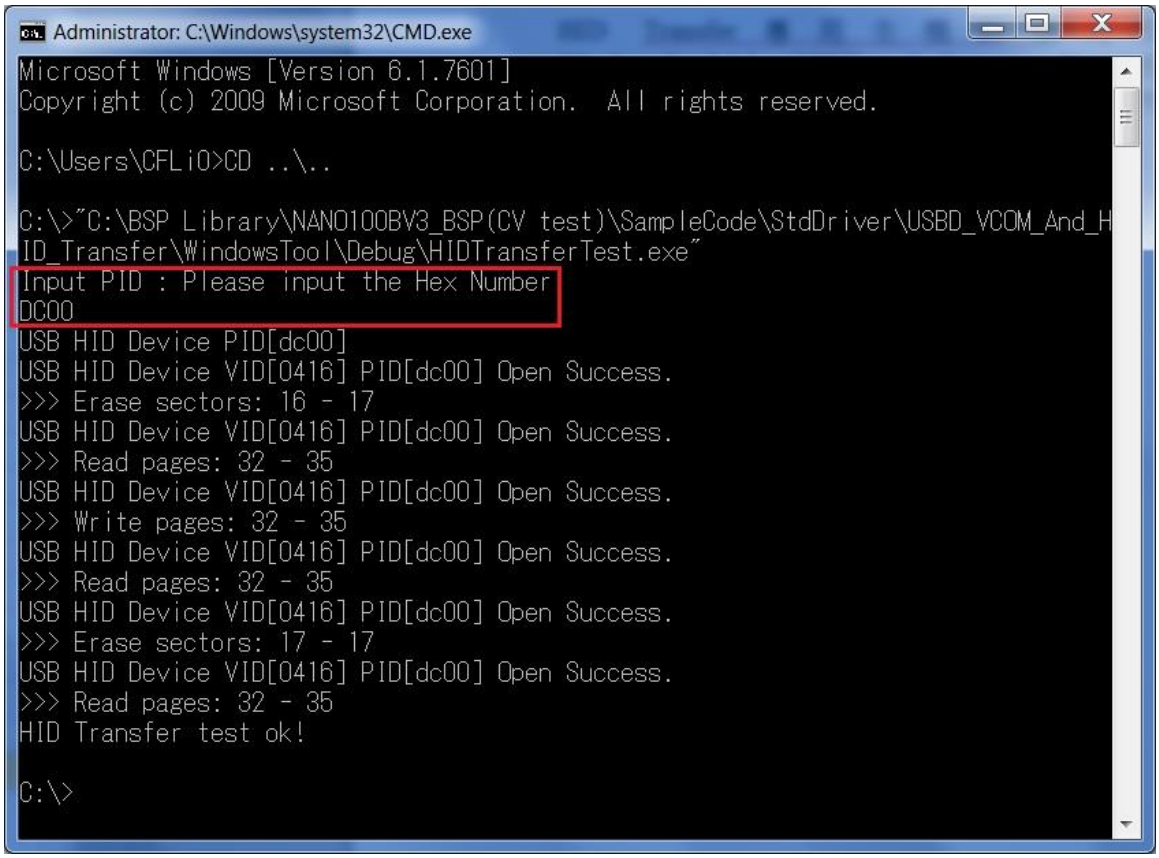


圖 30 HIDTransferTest.exe執行結果

Revision History

Date	Revision	Description
2015.10.30	1.00	1. 初次發布
2017.02.10	1.01	1. 更新 Audio + HID Transfer 範例中 bFunctionSubclassClass 之 IAD 欄位.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.